Standardaufgaben der Sekundarstufe I und II mit Maxima lösen

Version: 2. Januar 2013

Roland Stewen
stewen.rvk@gmx.de

Eike Schütze

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Wie benutzt man dieses Skript?

Dieses Skript ist nicht dazu gedacht, dass Sie es von vorne bis hinten durchlesen. Sondern Sie haben ein mathematisches Problem, welches Sie mit Maxima einem Computeralgebrasystem lösen wollen. Dazu suchen Sie im Inhaltsverzeichnis nach einer Beispielslösung. In der Beispielslösung wird auf den Eingabecode im Inhaltsverzeichnis verwiesen. Diesen können Sie kopieren und dann Ihren Bedürfnissen anpassen.

Maxima ist ein Computeralgebrasystem, welches heutzutage als Open-Source-Projekt weiterentwickelt wird. Es kann mit Buchstaben rechnen und sämtliche mathematischen Operationen durchführen wie Ableiten, Integrieren etc. Graphische Ausgaben werden an Gnuplot weitergereicht.

In diesem kleinen Skript sollen Ihnen Beispiele und kleine Kochrezepte zu Aufgaben aus dem Schulbereich gezeigt werden. Vieles finden Sie auch in der Hilfe, bzw. vieles weiterführende finden Sie in der Hilfe zu Wxmaxima. Viele interessante Aufgabenstellungen sind hier nicht aufgenommen, weil sie in der Schule nicht vorkommen. Auch finden manche Optionen hier keine Erwähnung. Gerade die Optionen sollten Sie nach kurzer Zeit unbedingt auch in der Hilfe bei Wxmaxima nachschauen.

Die pdf-Version dieser html-Dateien: maxima_in_beispielen.pdf

Verbesserungsvorschläge durch: Jan Müller.

Besonderer Dank gebührt Volker van Nek.

1.2 Einfache Kommandos und erste Bedienung

1.2.1 Anweisungen und Kommentare

Anweisungen werden mit einem Semikolon oder einem Dollarzeichen abgeschlossen. Bei einem Dollarzeichen gibt es am Ende keine Ausgabe.

Das Semikolon bietet Ihnen also eine Kontrollmöglichkeit. Wenn Sie eine neue Maxima-Funktion schreiben oder kleinere Eingaben haben, dann macht es Sinn die Ausgabe zu unterdrücken, weil in diesem speziellen Fall die Ausgabe recht unübersichtlich ist und keinen Kontrolleffekt hat.

```
(%i1) 3 * 4;
(%o1)                                  12
(%i2) 3 * 4$ /* Dies gibt keine Ausgabe */
```

Abbildung 1.1: Der Code ist hier: 197

Zwei Zahlen multiplizieren Sie mit einem Stern. Beachten Sie, dass Sie $2x$ in Maxima mit einem Nultiplikationszeichen eingeben müssen: `2 * x`.

Kommentare werden eingeschlossen durch einen Schrägstrich und einen Stern:
„/*„ Kommentar. „*/. Sie können an jeder Stelle einen Kommentar schreiben.

Wenn Sie mit **Wxmaxima** arbeiten müssen Sie die Anweisungen mit der Tastenkombination *shift return* abschließen. Nur durch drücken der Enter Taste erhalten Sie zwar eine neue Zeile aber keine Auswertung durch Maxima. Andererseits können Sie durch dieses sicherlich gewöhnungsbedürftige Verfahren Ihren Text - also Ihre Eingaben - übersichtlich gestalten.

```
(%i1) /* Summe */
  3 * 4  /* Kommentar 1 */
+ 2 * 3  /* Kommentar 2 */
;
(%o1)                                  18
```

Abbildung 1.2: Der Code ist hier: 198

Sie können in Maxima durch betätigen der Enter-Taste in einen Ausdruck Zeilenumbrüche einbauen. Dadurch werden manche Ausdrücke klarer strukturiert. Besonders bei eigenen Funktionsdefinitionen ist das sinnvoll.

Sie sollten alle Ergebnisse bzw. Ausdrücke mittels des Doppelpunktes einen Namen geben.

```
(%i1) a : 3 * 4$
(%i2) a;
(%o2)                                  12
```

Abbildung 1.3: Der Code ist hier: 199

Damit ist auch klar, dass der Doppelpunkt kein „Divisionszeichen“ mehr ist.

Durch den Aufruf von a wird hier das Ergebnis von $3 \cdot 4$ aufgerufen.

In Maxima werden die Ein- und Ausgabe bezeichnet. Somit können Sie natürlich auch mit Hilfe der Nummerierung auf die einzelnen Eingaben und Ausgaben zugreifen:

```
(%i1) a : 3 * 4;
(%o1)                                  12
(%i2) %i1 / 2;
                                     a : 12
(%o2)                                ------
                                       2
(%i3) ev (%i1 / 2);
(%o3)                                  6
(%i4) %o1 / 2;
(%o4)                                  6
(%i5) %  + 1;
(%o5)                                  7
```

Abbildung 1.4: Der Code ist hier: 200

In (%i2) wird der Ausdruck von (%i1) durch 2 geteilt. Sie können den Ausdruck durch **ev** (evaluate) auswerten lassen. Wenn Sie (%o1) genommen hätten, dann wäre ebenfalls 12 / 2 berechnet worden.

Wenn Sie nur ein %-Zeichen schreiben, dann wird der vorherige Ausdruck benutzt. Wenn Sie das allerdings oft machen, wird Ihre Rechnung unübersichtlich.

Die Empfehlung lautet, alle Ausdrücke mit einem treffendem Namen zu belegen und mit den Namen zu arbeiten!

Wenn Sie die Arbeit mit Maxima arbeiten wollen, dann tun Sie dies mit: quit(); Denken Sie an das Semikolon! Eine kurze Maxima-Session sieht z.B. so aus:

```
(%i1) 3 * 4;
(%o1)                                  12
(%i2) quit();
```

Abbildung 1.5: Der Code ist hier: 201

Sollten Sie mit Wxmaxima arbeiten, was empfehlenswert ist, dann ist das **quit**() nicht nötig.

Als Vorgriff auf die Funktionen sei hier schon erwähnt, dass Sie Funktionen nicht nur mit dem Doppelpunkt eingeben, sondern auch noch mit einem Gleichzeichen:

```
(%i1) f(x) := 2*x + 1;
(%o1)                          f(x) := 2 x + 1
(%i2) f(1);
(%o2)                                  3
(%i3) f(x), x = 1;
(%o3)                                  3
```

Abbildung 1.6: Der Code ist hier: 202

Dafür können Sie dann die gewohnte Schreibweise benutzen.

Man kann auch die Funktion aufschreiben und mit Komma abgetrennt die Werte der einzelnen Variablen (hier nur das x) angeben.

1.2.2 Grundoperationen mit Zahlen

Multiplikation

Multiplizieren können Sie mit dem Stern „*“ oder auch mit einem einfachen „.“. Bei Matrizen (und Vektoren) unterscheiden sich diese Multiplikationszeichen. Das Skalarprodukt und die normale Matrizenmultiplikation wird mit einem Punkt durchgeführt.

```
(%i1) 4*3;
(%o1)                                  12
```

Abbildung 1.7: Der Code ist hier: 203

Division

Die Division erfolgt durch einen Schrägstrich: „/“. Beachten Sie, dass Sie nicht den Doppelpunkt verwenden können, denn dies bedeutet eine Zuweisung.

```
(%i1) 4 / 2;
(%o1)                                   2
(%i2) 4 : 2;
assignment: cannot assign to 4
 -- an error. To debug this try: debugmode(true);
```

Abbildung 1.8: Der Code ist hier: 204

Ein Doppelpunkt generiert einen Fehler da die 2 nicht der 4 zugewiesen *(assign)* werden kann.

1.2.3 Variablen und Konstanten

Als Variablennamen dürfen Sie alles wählen, was nicht schon in Maxima belegt ist. Es ist sinnvoll möglichst Namen zu wählen, die der Bedeutung des Terms entsprechen.

Konstanten wie die Eulersche Zahl e, pi oder phi werden in Maxima einem Prozentzeichen vorangestellt.

```
(%i1) e^2, numer;
                                        2
(%o1)                                  e
(%i2) %e^2, numer;
(%o2)                          7.38905609893065
(%i3) exp (2);
                                         2
(%o3)                                  %e
```

Abbildung 1.9: Der Code ist hier: 205

In der ersten Zeile ist e eine ganz normale Variable, der keine Zahl zugewiesen wurde. Daher kann auch numer e^2 nicht zu einer Zahl auflösen.

1.2.4 Logarithmus

Sie wollen den Logarithmus einer positiven Zahl zur Basis 2 berechnen.

```
(%i1) log(%e);
(%o1)                                  1
(%i2) log(8) / log(2), numer;
(%o2)                                3.0
```

Abbildung 1.10: Der Code ist hier: 206

log (*Zahl*) gibt Ihnen den natürlichen Logarithmus der Zahl, d. h. den Logarithmus zur Euler-Zahl e. Die Konstante e wird in Maxima mit einem Prozentzeichen geschrieben: %e. **log** (*x*) entspricht ln(x).

So ist **log** (*e*) naturgemäß 1, denn es gilt: $e^1 = 1$.

Wenn Sie den Logarithmus zur Basis 2 der Zahl 8 errechnen wollen, müssen Sie durch den Logarithmus der Basis teilen:

$$\log_2(8) = \frac{\ln(8)}{\ln(2)}$$

1.3 Zuweisungen und Funktionen

1.3.1 Einfache Funktionszuweisungen und Auswertungen

Ausdrücke werden mit einem Doppelpunkt zugewiesen. Eine Funktion dagegen wird durch := deklariert. Bei einer Funktion können Sie einzelne Werte in Klammern übergeben, so wie Sie es in normaler Schreibweise von Funktionen gewohnt sind, oder Sie schreiben die Werte der Variablen daneben.

Bei Ausdrücken schreiben Sie den Wert der einzelnen Variablen daneben.

```
(%i1) /* Definition */
a : 2 * x;        /* Ausdruck */
(%o1)                                  2 x
(%i2) f(x) := 2 * x;    /* Funktion */
(%o2)                               f(x) := 2 x
(%i3) f : 2 * x + 1;    /* Ausdruck */
(%o3)                                2 x + 1
(%i4) /* Auswertungen */
(%i4) a, x = 1;
(%o4)                                  2
(%i5) f(1);
(%o5)                                  2
(%i6) f(x), x = 1;
(%o6)                                  2
(%i7) f, x = 1;         /* Dies ist nicht f(x) */
(%o7)                                  3
```

Abbildung 1.11: Der Code ist hier: 207

Sie können f als Ausdruck benennen und auch als Funktion festlegen. Dies sind aber zwei unterschiedliche Objekte!

1.3.2 Funktionszuweisungen mit ” bzw. define und ev

Sie können Funktionen auch durch Funktionen von Maxima definieren. Dazu benötigen Sie in der Regel zwei Apostrophe. Sie können aber auch mit **define** (*Funktion, Anweisungen*) oder **ev** (*Anweisungen*) arbeiten. Im folgenden Beispiel sehen Sie die Anwendung der beiden Apostrophe und von **ev** (*Anweisungen*). Bei der Verwendung von **ev** wie evaluate wird die Funktion bei jedem Aufruf neu ausgewertet. Sollte sich also zwischenzeitlich etwas geändert haben, so hat das Seiteneffekte.

```
(%i1) f(x) := x^2$
(%i2) df1(x) := ''( diff ( f(x), x ) )$
(%i3) define ( df2(x), diff ( f(x), x ) )$
(%i4) df3(x) := ev ( diff ( f(x), x ) )$
(%i5) df1(x);
(%o5)                                2 x
(%i6) df2(x);
(%o6)                                2 x
(%i7) df3(x);
(%o7)                                2 x
(%i8) f(x) := x^3$
(%i9) df1(x);
(%o9)                                2 x
(%i10) df2(x);
(%o10)                               2 x
(%i11) df3(x);
                                       2
(%o11)                              3 x
```

Abbildung 1.12: Der Code ist hier: 208

Die Funktionen df1, df2 und df3 werden definiert. Die Definition durch **define** und diejenige durch Hochkommata verändern sich nicht, wenn f verändert wird. df3, welche mit Hilfe von **ev** definiert wurde ändert sich dagegen, wenn sich f ändert.

1.4 Hilfe

1.4.1 Befehlsname suchen

Sie suchen einen Befehlsnamen und wissen nur noch einen Teil des Namens.

```
(%i1) apropos ("num");
(%o1) [%enumer, %e_to_numlog, %rnum, %rnum_list, alphanumericp, bignum,
ecm_number_of_curves, filenum, fixed_num_args_function, fixnum, floatnum,
floatnump, flonum, gensumnum, linenum, lognumer, maxpsifracnum, nonumfactor,
num, number, numberp, numer, numerval, numer_pbranch, numfactor, nummod,
num_distinct_partitions, num_partitions, primep_number_of_tests, ratnum,
ratnumer, ratnump, signum, storenum, subnumsimp, tr_numer, ttyintnum,
variable_num_args_function, num, signum]
```

Abbildung 1.13: Der Code ist hier: 209

Sie bekommen mit **apropos** alle Befehle, welche „num“ im Namen haben. Denken Sie daran, den Teil des Namens in Anführungszeichen zu schreiben.

Wenn Sie **apropos (“ “);** eingeben, erhalten Sie alle Maxima Befehle.

Alternativ zu **apropos** können Sie auch mit **describe** (*“Befehl“*) arbeiten. Wenn Sie zwei Fragezeichen benutzen: `??` *solve* (ohne Anführungszeichen), dann wird inexakt gesucht und alle Funktionspakete aufgelistet, in denen diese Zeichenkette vorkommt.

Wenn Sie mit **Wxmaxima** arbeiten können Sie stattdessen besser den Index bzw. die Suche benutzen.

1.4.2 Beispiele zu einem Befehl

example (Befehl) gibt Ihnen - soweit vorhanden - Beispiele. Hier werden jetzt Beispiele zu dem Befehl **append** geliefert.

```
(%i1) example (append);
(%i2) append([x+y,0,-3.2],[2.5E+20,x])
(%o2)                    [y + x, 0, - 3.2, 2.5E+20, x]
(%o2)                                done
```

Abbildung 1.14: Der Code ist hier: 210

Wenn Sie mit Wxmaxima arbeiten gibt es zu jedem Befehl eine Dokumentation. Sie können auch nach Befehlen suchen lassen, selbst wenn Sie nur Teile des Befehls kennen.

1.5 Batch-Dateien

Besonders angenehm ist es, dass Sie die Anweisungen für Maxima in eine Datei schreiben können und Maxima dann diese Datei auswertet. Dazu müssen Sie Maxima mit der Option -b aufrufen:
maxima -b meineDatei
Geben Sie Ihren Dateien die Endung „.mac“.

Sie können auch Maxima oder Wxmaxima wie gewohnt starten und dann mit **batchload** (*“dateiname“*) arbeiten. So können Sie Funktionen, welche Sie geschrieben haben immer wieder laden. Als Lehrer können Sie Funktionen zur Verfügung stellen wie z. B. das Vektorprodukt, ohne dass die Schüler die genaue Ausführung der Funktion kennen.

Kapitel 2

Zahlen

2.1 Nachkommastellen

Wie kann man die Nachkommastellen der Zahlen verändern?

Maxima rechnet standardmäßig mit 16 Stellen Genauigkeit. Da Maxima anders als bei einem Taschenrechner nicht mit den Zwischenergebnissen sondern die umgeformten Terme benutzt reicht das in der Regel auch aus.

```
(%i1) 123456789123456789123456678
*
123123123123123123123123123;
(%o1)        15200385447632817880065236615430245182997812750565394
(%i2) bruch : 1/3;
                                      1
(%o2)                                 -
                                      3
(%i3) bruch, float;
(%o3)                          .3333333333333333
(%i4) fpprec : 100;
(%o4)                                100
(%i5) bfloat (bruch);
(%o5)
3.3333333333333333333333333333333333333333333333333333333333333333333333333333333333\
33333333333333333333333333333b-1
(%i6) fpprintprec : 2;
(%o6)                                 2
(%i7) 5 / 3, numer;
(%o7)                                1.7
(%i8) fpprintprec : 20; /* ergibt trotzdem nur 16 Nachkommastellen */
(%o8)                                20
(%i9) 5 / 3, numer;
(%o9)                          1.666666666666667
```

Abbildung 2.1: Der Code ist hier: 211

In Maxima können Sie beliebig große Zahlen eingeben. Die Genauigkeit ist jedoch festgeschrieben (bei **float**-Zahlen, dass sind Gleitkommazahlen) auf 16 Stellen. Wenn Sie weniger Zahlen angezeigt bekommen wollen, dann können Sie **fpprintprec** verwenden.

Wenn Sie davon abweichend mit mehr oder weniger Genauigkeit rechnen wollen, benötigen Sie **bfloat**-Zahlen. **bfloat** steht für big float. Die Nachkommastellen werden mit **fpprec : 32** verändert. (Diese Angabe veränder aber nicht die normalen Nachkommazahlen). fpprec steht für floating point precision.

bfloat (*Zahl*) wandelt eine Dezimalzahl in eine **bfloat**-Zahl um.

2.2 Brüche

2.2.1 Bruch als Dezimalzahl

Sie wollen einen Bruch in eine Dezimalzahl umwandeln.

```
(%i1) a : 2/3;
                                        2
(%o1)                                   -
                                        3
(%i2) a, numer : true;
(%o2)                          .6666666666666666
(%i3) a, numer;
(%o3)                          .6666666666666666
```

Abbildung 2.2: Der Code ist hier: 212

Mit **numer : true** oder kurz ein numer mit Komma abgetrennt wird der Bruch als Dezimalzahl bestimmt.

2.2.2 Dezimalzahl als Bruch

Mit **rat** (*Zahl*) können Sie eine Dezimalzahl in einen Bruch umwandeln.

```
(%i1) a : 0.2;
(%o1)                                  0.2
(%i2) rat (a);
rat: replaced 0.2 by 1/5 = 0.2
                                        1
(%o2)/R/                                -
                                        5
(%i3) ratprint : false;
(%o3)                                 false
(%i4) rat (a);
                                        1
(%o4)/R/                                -
                                        5
```

Abbildung 2.3: Der Code ist hier: 213

Diese Rückmeldung können Sie durch **ratprint : false** unterdrücken.

2.2.3 Zähler eines Bruches

Sie wollen den Zähler eines Bruches erhalten.

Damit können Sie den Zähler ausgeben oder an eine Variable übergeben.

```
(%i1) a : 3 / 4;
                                        3
(%o1)                                   -
                                        4
(%i2) ratnumer (a);
(%o2)/R/                                3
(%i3) ratnumer (0.2);
rat: replaced 0.2 by 1/5 = 0.2
(%o3)/R/                                1
```

Abbildung 2.4: Der Code ist hier: 214

num (*Bruch*) gibt den Zähler des Bruches zurück.

2.2.4 Nenner eines Bruches

Sie wollen den Nenner eines Bruches erhalten.

```
(%i1) denom (3/4);
(%o1)                                   4
```

Abbildung 2.5: Der Code ist hier: 215

denom (*Zahl*) gibt den Nenner des Bruches zurück.

2.2.5 Umwandlung eines Bruches in eine Dezimalzahl

Sie haben einen Bruch und wollen diesen in eine Dezimalzahl umwandeln.

```
(%i1) rat (0.5 * x);
rat: replaced 0.5 by 1/2 = 0.5
                                        x
(%o1)/R/                                -
                                        2
(%i2) rat (0.5 * x), keepfloat;
(%o2)/R/                              0.5 x
(%i3) solve (2.0 * x = 1, x), keepfloat;
rat: replaced 2.0 by 2/1 = 2.0
                                          1
(%o3)                                [x = -]
                                          2
(%i4) ratprint : false;
(%o4)                                 false
(%i5) solve (2.0 * x = 1, x), keepfloat;
                                          1
(%o5)                                [x = -]
                                          2
```

Abbildung 2.6: Der Code ist hier: 216

Wenn Sie sicherstellen wollen, keinen Bruch zu erhalten sondern nur Dezimalzahlen, dann geben Sie hinter der Anweisung mit einem Komma abgetrennt die Anweisung **keepfloat**

Mit **solve** (*Gleichung, Var*) lösen Sie Gleichungen. **solve** ignoriert **keepfloat**.

ratprint : false unterdrückt die Rückmeldung zur Umwandlung.

Kapitel 3

Terme

3.1 Äquivalenz von Termen

Sie wollen entscheiden ob zwei Terme äquivalent sind.

Damit können Sie nachschauen, ob Sie richtig umgeformt haben.

$$3(2+x)-4$$

und

$$6+3x-4$$

```
(%i1) is ( equal (
        3 * ( 2 + x) - 4,
        6 + 3 * x - 4
        )
    );
(%o1)                                true
```

Abbildung 3.1: Der Code ist hier: 217

is (equal ((Ausdruck, Ausdruck) gibt „true“ zurück, wenn die Ausdrücke gleich sind. Wenn Sie nicht gleich sind, dann erhalten Sie „false“.

3.2 Term ausmultiplizieren

Sie wollen einen Term ausmultipliziert und zusammengefasst haben.

$$3(x+2)+4$$

```
(%i1) expand ( 3 * (x + 2) + 4 );
(%o1)                                3 x + 10
```

Abbildung 3.2: Der Code ist hier: 218

expand (*Ausdruck*) expandiert einen Term.

Wenn Sie einen Term mit einem Logarithmus oder einer Wurzel haben, dann können Sie mit **logexpand** bzw. **radexpand** Feinheiten einstellen. Konsultieren Sie dazu die Hilfe.

3.3 Terme Faktorisieren

Sie möchten einen Term Faktorisieren.

```
(%i1) factor ( x^2 + 2 * x );
(%o1)                                x (x + 2)
(%i2) e : %e$
(%i3) factor ( e^x * x + 4 * e^x );
                                              x
(%o3)                                (x + 4) %e
```

Abbildung 3.3: Der Code ist hier: 219

Zur leichteren Eingabe ist hier e als %e definiert. In Maxima ist %e die Eulersche Zahl.

factor (*Ausdruck*) faktorisiert einen Term. Dabei wird das maximal Mögliche ausgeklammert.

3.4 Werte in Terme einsetzen

Sie haben einen Term und wollen Werte für die einzelnen Buchstaben einsetzen. Die kinetische Energie beträgt:

$$E_{\text{kin}} = \frac{1}{2}\, m\, v^2$$

Sie wollen die kinetische Energie ausrechnen für eine Masse von 2 kg und eine Geschwindigkeit von 3 m/s.

```
(%i1) E : 1/2 * m * v^2;
                                             2
                                          m v
(%o1)                                     ----
                                           2
(%i2) E, m = 2, v = 3;
(%o2)                                      9
```

Abbildung 3.4: Der Code ist hier: 220

Schreiben Sie den Term auf und trennen Sie mit Kommata die Werte für die einzelnen Buchstaben. Maxima ersetzt dann die Buchstaben oder Variablen durch die einzelnen Werte und vereinfacht dann den Ausdruck.

Kapitel 4

Gleichungen

4.1 Einfache Gleichungen

4.1.1 Lösen einer Gleichung

Sie haben eine Gleichung und wollen diese lösen. Dazu brauchen Sie **solve** (*Gleichung, Variable*)

Sie können **solve** verschieden aufrufen:

1. **solve** (*Gleichung*)
2. **solve** (*Gleichung, Variable*)
3. **solve** (*Gleichung Variablenliste*)

In der Regel reicht Ihnen die Ausgabe von **solve**. Sie können mit dieser Ausgabe auch Funktionswerte berechnen.

Wenn Sie jedoch die Zahl explizit haben wollen, geht das auch und ist weiter unten der Vollständigkeit halber beschrieben. Aber in der Regel reichen die ersten beiden Zeilen auch der kommenden Beispiele.

```
(%i1) solve (10*x + 5 = 15);
(%o1)                                [x = 1]
(%i2) lsg : solve (10*x + 5 = 15);
(%o2)                                [x = 1]
(%i3) lsg;
(%o3)                                [x = 1]
(%i4) 10 * x + 5, lsg;
(%o4)                                  15
```

Abbildung 4.1: Der Code ist hier: 221

Beachten Sie, dass Sie keine Ausgabe erhalten, wenn Sie die Zeile mit $ abschließen:

Wenn Sie mit den Lösungen weiterarbeiten wollen, sollten Sie die Lösung einem treffenden Namen zuweisen.

4.1.2 Lösen einer Exponentialgleichung

Sie haben eine Exponentialgleichung

$$2^{2x} = 4$$

```
(%i1) a : solve ( 2^(2*x) = 4 );
                                    log(- 2)
(%o1)                          [x = --------, x = 1]
                                     log(2)
(%i2) float (a);
(%o2) [x = 1.442695040888963 (3.141592653589793 %i + .6931471805599453),
                                                                      x =
1.0]
(%i3) sublist (float(a), lambda ( [x], freeof(%i, x) ) );
(%o3)                                [x = 1.0]
(%i4) sublist (a, lambda ( [x], imagpart ( rhs(x) ) = 0 ) );
(%o4)                                 [x = 1]
```

Abbildung 4.2: Der Code ist hier: 222

Sie übergeben diese Gleichung jeweils an **solve** (*Gleichung, Variable*). Maxima gibt Ihnen nicht nur Lösungen im reellen Zahlenbereich sondern auch die komplexen Zahlen, welche die Gleichung lösen. Um die reellen Zahlen zu erhalten können Sie z. B. folgendermaßen vorgehen:

1. Sie erstellen eine Liste der Zahlen, welche kein „i“ enthalten. **freeof** (*var1, var2, . . . , Ausdruck*) ist „wahr“, wenn keine der Variablen im Ausdruck vorhanden ist. **freeof** (*%i, x*) ist also „wahr“, wenn in der Zahl, welche der Variablen x übergeben wird kein %i enthalten ist, sie also reell ist.

   sublist (*Liste1, Funktion*) erstellt eine neue Liste mit den Elementen aus Liste1, für die die Funktion den Wert „wahr“ ergibt. Die Funktion kann dann natürlich nur eine Funktion mit einem Argument sein.

   Hier wird als Funktion die anonyme Funktion lambda gewählt. Jedes Element der Liste wird an Lambda der Variablen x übergeben. **lambda** (*[lokale Variablenliste], Anweisungen*) überprüft für jede Zahl aus der Lösungsliste, ob es ein i enthält.

2. Sie erstellen eine Liste der Zahlen bei denen der imaginäre Teil der Zahl null ist.

   rhs (*Gleichung*) gibt die rechte Seite der Gleichung zurück. Das sind in unserem Fall die Lösungen der Gleichung.

   imagpart (*Ausdruck*) gibt den imaginären Anteil des Ausdrucks zurück.

   Die Funktion **lambda** gibt den Wert „wahr“ zurück, wenn der imaginäre Anteil des Ausdrucks gleich null ist.

   sublist (*Liste1, Funktion*) erstellt dann eine Unterliste aus der Liste a, bei deren Elemente der imaginäre Anteil null ist.

4.1.3 Gleichungssysteme und Ubersicht

Sie haben mehrere Gleichungen und wollen nicht die Übersicht verlieren.

Schreiben Sie die Gleichung oder die Gleichungen daher vorher separat auf:

```
(%i1) gl1 : 10 * x + 5 = 15$
(%i2) lsg : solve (gl);
(%o2)                                [gl = 0]
(%i3) /* Mehrere Gleichungen */
gl2 : 10 * x - 5 = y$
(%i4) gl3 : 5 * x + 10 = y$
(%i5) lsg : solve ([gl2, gl3], [x, y]);
(%o5)                            [[x = 3, y = 25]]
```

Abbildung 4.3: Der Code ist hier: 223

Durch das $-Zeichen haben Sie keine Bestätigung. Hier machen wir dadurch lediglich die Ausgabe besser lesbar.

Solange Sie nur eine Variable haben, müssen Sie sie nicht explizit **solve** mitteilen.

Wenn Sie zwei Gleichungen lösen lassen wollen, dann übergeben Sie eine Liste der Gleichungen und eine weitere Liste mit den Variablen. **solve** (*Liste von Gleichungen, Liste der Variablen*) löst dann die Gleichungen.

4.1.4 Gleichungen lösen und die Zahlwerte erhalten

```
(%i1) gl1 : y = 2 * x + 1$
(%i2) gl2 : 2 * y - 5 * x = -1$
(%i3) /*
  Der Rest des Codes dient dazu, um
  mit den einzelnen Werten
  weiterrechnen zu koennen.
*/
lsg : solve ([gl1, gl2], [x, y]);
(%o3)                            [[x = 3, y = 7]]
(%i4) lsg [1][1];
(%o4)                                 x = 3
(%i5) rhs (lsg[1][1]);
(%o5)                                   3
(%i6) rhs (lsg[1][2]);
(%o6)                                   7
```

Abbildung 4.4: Der Code ist hier: 224

Beachten Sie, dass Sie bei **solve** die Gleichungen in einer Liste angeben und danach die Variablenliste.

Hier ist die Ausgabe eine Liste von einer Liste. Wenn Sie mit der Lösung weiterarbeiten wollen, dann müssen Sie auf das erste Element der äußeren Liste zugreifen, welches wiederum eine 1-elementige Liste ist.

Wenn Sie nur die Zahl haben wollen, dann können Sie mit **rhs** arbeiten. lsg[1] ist das erste Element der äußeren Liste und lsg[1][1] ergibt dann die erste Lösung. lsg[1][2] ist dann das zweite Element der Lösung. Die Ausgabe von **rhs** (*lsg[1][1]*); ergibt also gerade die Zahl, welche die erste Lösung ergibt.

4.1.5 Lösung eines Gleichungssystems als Zahlen

Sie wollen eine Liste erstellen mit den jeweiligen Lösungen als Zahl. Also nicht mehr eine Liste deren Elemente jeweils x=3 oder x=4 enthalten.

```
(%i1) gl1 : y = 2 * x + 1$
(%i2) gl2 : 2 * y - 5 * x = -1$
(%i3) lsg : solve ([gl1, gl2], [x, y]);
(%o3)                          [[x = 3, y = 7]]
(%i4) loesungsliste : map (rhs, (lsg[1]));
(%o4)                                [3, 7]
(%i5) [x1, x2] : map (rhs, (lsg[1]));
(%o5)                                [3, 7]
(%i6) x1;
(%o6)                                   3
(%i7) gl1, lsg[1][1];  /* y-Wert */
(%o7)                                y = 7
(%i8) gl1, x = x1;     /* Alternativ */
(%o8)                                y = 7
```

Abbildung 4.5: Der Code ist hier: 225

Arbeiten Sie mit **map** (*Funktion, Liste*). Die Funktion wird auf jedes Element der Liste angewendet. Hier beinhaltet die Liste lsg[1] gerade die einzelnen Lösungen aus solve in der Form: x = Zahl. Nun wenden Sie auf jedes Element der Liste die Funktion **rhs** (*Gleichung*) an und erhalten dann eine neue Liste mit derselben Anzahl wie die übergebene Liste (lsg[1]) mit den jeweiligen Lösungen.

4.1.6 Schnittpunkt zweier Funktionen bestimmen

Den Schnittpunkt von zwei Funktionen bestimmen Sie folgendermaßen:

```
(%i1) f(x) := 2*x + 1;
(%o1)                          f(x) := 2 x + 1
(%i2) g(x) := 3*x - 1;
(%o2)                          g(x) := 3 x - 1
(%i3) lsg : solve (f(x) = g(x));
(%o3)                                [x = 2]
(%i4) /* Wie benutzt man diese Ergebnisse weiter? */
f(x), x = 2;
(%o4)                                   5
(%i5) f(x), lsg[1];
(%o5)                                   5
```

Abbildung 4.6: Der Code ist hier: 226

Sie können auch zwei Funktionen bestimmen und diese dann gleichsetzen. Die Lösung wird an lsg übergeben. Wenn Sie noch den y-Wert des Schnittpunktes berechnen wollen, dann ist die einfache Möglichkeit, dass Sie neben f(x), mit Komma abgetrennt, angeben wie groß x sein soll.

4.1.7 Lösung einer quadratischen Gleichung

Sie haben eine quadratische Gleichung und möchten die Lösungen erhalten.

```
(%i1) gl1 : 2 * t^2 - 8 * t + 6 = 0$
(%i2) lsg : solve (gl1);
(%o2)                          [t = 3, t = 1]
(%i3) /* Wenn Sie das Ergebnis weiterverwenden moechten */
lsgliste : map (rhs, lsg);
(%o3)                               [3, 1]
(%i4) lsgliste[1];
(%o4)                                  3
(%i5) lsgliste[2];
(%o5)                                  1
```

Abbildung 4.7: Der Code ist hier: 227

lsg ist eine Liste mit Lösungen. Jedes Element dieser Liste ist von der Form: x = Zahl.

map (*Funktion, Liste*) gibt eine Liste zurück, welche dieselbe Anzahl von Elementen wie die übergebene Liste hat. Auf jedes Element der übergebenen Liste (lsg) wird die Funktion **(rhs)** angewendet.

rhs (*Gleichung*) gibt Ihnen die rechte Seite der Gleichung.

4.2 Gleichung selber umformen

Sie haben eine Gleichung und wollen Sie umformen.

$$3(x+4)+2=17$$

Die Umformungen können Sie auf verschiedene Weise durchführen.

1. Sie schreiben eine Gleichung auf, umklammern diese Gleichung (mit eckigen oder runden Klammern) und vollziehen dann den Umformungsschrit. Z. B. von beiden Seiten 2 subtrahieren.

2. Sie benennen die Gleichung und jede neu entstandene Gleichung mit einem Namen. Z. B. gl1, gl2 usw. Und Sie vollziehen die Umformungsschritte dann mit jedem Namen.

   Dieses Verfahren hat den Vorteil, dass Sie die Struktur der Umformung erkennen können.

3. Alternativ können Sie immer die vorhergehende Gleichung nehmen, indem Sie die Gleichung nicht selber benennen, sondern den von Maxima vorgegebenen Namen wie (%o1), (%o2) usw. verwenden. Das wird aber sehr schnell unübersichtlich. Diese Alternative ist also nur für wenige Umformungen interessant.

4.2.1 Gleichungen in Klammern einschliessen

```
(%i1) [3 * (x + 4) + 2 = 17] - 2;
(%o1)                          [3 (x + 4) = 15]
(%i2) [3 * (x + 4) = 15] / 3;
(%o2)                             [x + 4 = 5]
(%i3) [x + 4 = 5] -4;
(%o3)                               [x = 1]
```

Abbildung 4.8: Der Code ist hier: 228

Sie können die Gleichung in rechteckige und runde Klammern einschliessen. Hier sind nur die rechteckigen gezeigt.

Der Nachteil dieser Methode ist, dass Sie die Gleichung neu schreiben bzw. kopieren müssen und Sie auf das Ergebnis nur über den Maxima Namen (%o3) zugreifen können.

4.2.2 Gleichungen mit Namen

```
(%i1) gl1: 3 * (x + 4) + 2 = 17;
(%o1)                         3 (x + 4) + 2 = 17
(%i2) gl2 : gl1 - 2;
(%o2)                           3 (x + 4) = 15
(%i3) gl3 : gl2 / 3;
(%o3)                              x + 4 = 5
(%i4) gl4 : gl3 - 4;
(%o4)                                x = 1
```

Abbildung 4.9: Der Code ist hier: 229

Hier wird die Struktur der Gleichungslösung besonders deutlich. Besonders, wenn Sie nur den Code abspeichern und ihn dann in WxMaxima hineinkopieren oder als batchdatei ausführen lassen.

4.2.3 Gleichungen mit Maximas Namen

```
(%i1) 3 * (x + 4) + 2 = 17;
(%o1)                         3 (x + 4) + 2 = 17
(%i2) %o1 - 2;
(%o2)                           3 (x + 4) = 15
(%i3) %o2 / 3;
(%o3)                              x + 4 = 5
(%i4) %o3 - 4;
(%o4)                                x = 1
```

Abbildung 4.10: Der Code ist hier: 230

Diese Möglichkeit geht auch.

4.3 Gleichungen mit mehreren Lösungen

Sie suchen den Schnittpunkt zweier Geraden, welche aufeinander liegen.

```
(%i1) f(x) := 2*x + 1;
(%o1)                          f(x) := 2 x + 1
(%i2) g(x) := 2*x + 1;
(%o2)                          g(x) := 2 x + 1
(%i3) solve ( f(x) = g(x) );
(%o3)                                all
```

Abbildung 4.11: Der Code ist hier: 231

Wenn Sie mehrere Gleichungen haben und Sie genauere Informationen über den Lösungsraum haben wollen, schreiben Sie die Gleichungen mit Variablen auf. Also y statt f(x).

```
(%i1) gl1: y = 2*x + 1;
(%o1)                              y = 2 x + 1
(%i2) gl2: y = 2*x + 1;
(%o2)                              y = 2 x + 1
(%i3) lsg : solve ( [gl1, gl2], [x, y]);
solve: dependent equations eliminated: (2)
                                  %r1 - 1
(%o3)                       [[x = -------, y = %r1]]
                                    2
(%i4) lsg : subst (t, %rnum_list[1], lsg);
                                  t - 1
(%o4)                       [[x = -----, y = t]]
                                    2
(%i5) /* Aufloesen nach y */
lsg2 : solve ( [gl1, gl2], [y, x]);
solve: dependent equations eliminated: (2)
(%o5)                      [[y = 2 %r2 + 1, x = %r2]]
(%i6) lsg2 : subst (t, %rnum_list[1], lsg2);
(%o6)                        [[y = 2 t + 1, x = t]]
```

Abbildung 4.12: Der Code ist hier: 232

Bei diesem Ergebnis ist r1 eine freie Variable. Je nachdem wie Sie y (bzw. r1) wählen, müssen sie x wählen.

In %rnum_list sind die Variablennamen der Lösung gespeichert.

Statt %r1 können Sie auch als Parameter t wählen. Dazu müssen Sie in der Lösung lsg %r1 durch t ersetzen bzw. in lsg den 1. Variablennamen, welcher in **%rnum_list** gespeichert ist, durch t ersetzen. (Siehe auch: 12.3

Zur Substitution von %r1 durch t in lsg benutzen Sie **subst** (*a*, *b*, *c*). In dem Ausdruck c wird jedes Vorkommen von b durch a ersetzt. **subst** (*t*, *%rnum_list[1]*, *lsg*) ersetzt in lsg jedes %r1 durch ein t. (Sie können natürlich auch einfach **subst** (*t*, *%r1*, *lsg*) schreiben.)

Es wird aufgelöst nach der letzten Variablen (oder den letzten Variablen: soweit wie eben möglich). Wenn Sie statt **[x, y] [y, x]** schreiben, wird nach x aufgelöst.

4.4 Wurzelgleichungen

Sie haben eine Wurzelgleichung und wollen diese lösen.

$$\sqrt{x^2-1} = \sqrt{2x+1}$$

Das normale **solve** (*Variablenliste, Gleichungsliste*) löst Ihnen kompliziertere Wurzelgleichungen nicht.

```
(%i1) solve(sqrt(x^2-1) = sqrt(2*x+1), x); /* Bietet keine adaequate Loesung
*/
                                                    2
(%o1)                    [sqrt(2 x + 1) = sqrt(x  - 1)]
(%i2) load(to_poly_solver)$
(%i3) n : to_poly_solve(sqrt(x^2-1) = sqrt(2*x+1), x);
(%o3)             %union([x = 1 - sqrt(3)], [x = sqrt(3) + 1])
(%i4) first (n);
(%o4)                           [x = 1 - sqrt(3)]
(%i5) second (n);
(%o5)                           [x = sqrt(3) + 1]
```

Abbildung 4.13: Der Code ist hier: 233

solve (*Variablenliste, Gleichungsliste*) löst Ihnen die Gleichung nicht zu Ihrer Zufriedenheit.

Laden Sie das Paket mit **load** (*to_poly_solver*).

to_poly_solve (*Gleichung, Variable*) löst Ihnen die Gleichung. Der Rückgabewert ist eine Menge von Lösungen. Auf die Lösungen können Sie mit **first** (*Liste*)[1] bzw. **second** (*Liste*) usw. zugreifen.

[1] **first** können Sie auf Mengen und Listen anwenden.

Kapitel 5

Differenzialrechnung

5.1 Ableitungen einer Funktion

Die Ableitungen einer Funktion können Sie schnell bestimmen lassen. Ebenfalls können Sie auch Werte berechnen lassen:

```
(%i1) f(x) := x^3;
                                        3
(%o1)                              f(x) := x
(%i2) df : diff ( f(x), x);
                                         2
(%o2)                                 3 x
(%i3) df, x = 3;
(%o3)                                 27
(%i4) df : diff ( f(x), x, 2);
(%o4)                                 6 x
```

Abbildung 5.1: Der Code ist hier: 234

diff (*Ausdruck, Variable*) bzw. **diff** (*Ausdruck, Variable, Anzahl der Ableitungen*) leitet die Funktion ab. Wenn Sie höhere Ableitungen haben wollen, dann müssen Sie dies als drittes Argument angeben.

5.2 Ableitungsfunktion erstellen

Wenn Sie die Ableitung in eine Funktion umwandeln wollen, die Sie gegebenenfalls auch zeichnen lassen können, müssen Sie anders vorgehen. Sie müssen gewährleisten, dass erst differenziert wird und danach für x der entsprechende Wert (hier die 2) eingesetzt wird.

```
(%i1) f(x) := x^2;
                                          2
(%o1)                              f(x) := x
(%i2) df(x) : diff( f(x), x);
assignment: cannot assign to df(x)
 -- an error. To debug this try: debugmode(true);
(%i3) df(2);
(%o3)                                 df(2)
(%i4) df(x) := ''( diff (f(x), x) );
(%o4)                              df(x) := 2 x
(%i5) df(2);
(%o5)                                   4
(%i6) define ( df(x), diff ( f(x), x ) );
(%o6)                              df(x) := 2 x
(%i7) df(2);
(%o7)                                   4
```

Abbildung 5.2: Der Code ist hier: 235

diff (*Ausdruck, Variable*) bzw. **diff** (*Ausdruck, Variable, Anzahl der Ableitungen*) leitet die Funktion ab.

diff () erstellt Ihnen aber keine Funktion, sondern nur einen Ausdruck. Wenn Sie in dem Ausdruck einen konkreten Wert für x angeben wollen, geht dies nur darüber, dass Sie hinter dem Ausdruck mit einem Komma abgetrennt: x = 2 angeben.

Sie bekommen auf zweierlei Art eine Funktion:

1. Sie arbeiten mit zwei Hochkommatas und umschliessen **diff** mit einer Klammer.
2. Sie definieren eine Funktion mit Hilfe von **define** (*Funktion, Ausdruck*). df(x) ist die zu definierende Funktion und **diff**(f(x), x) ist der Ausdruck.

5.3 Finden einer Stammfunktion und bestimmtes Integral

Sie suchen eine Stammfunktion zu einer gegebenen Funktion f.

```
(%i1) integrate (x^2, x);
                                         3
                                        x
(%o1)                                   --
                                        3
(%i2) integrate (x^2, x, 1, 2);
                                         7
(%o2)                                    -
                                         3
```

Abbildung 5.3: Der Code ist hier: 236

Sie können mit **integrate** (*Funktion, Variable*) eine Stammfunktion erstellen. Sie erhalten ein bestimmtes Integral, wenn Sie das Intervall mit Kommata getrennt eingeben: **integrate** (*Funktion, Variable, 1, 2*)

5.4 Extremstellen eines Polynoms

Aufgabe

Bestimmen Sie die Extrempunkte der Funktion f.

$$f(x) = 4x^5 - 15x^4 - 80x^3 + 200x^2 + 960x$$

Lösung

```
(%i1) f(x) := x^3 - 12 * x^2 + 36 * x;
                                        3       2
(%o1)                        f(x) := x  - 12 x  + 36 x
(%i2) df : diff( f(x), x);
                                      2
(%o2)                               3 x  - 24 x + 36
(%i3) ddf : diff( f(x), x, 2);
(%o3)                                  6 x - 24
(%i4) lnst : solve(df = 0, x); /* Nullstellen der 1. Ableitung */
(%o4)                               [x = 6, x = 2]
(%i5) /* Einsetzen in die 2. Ableitung */
(%i5) ddf, x = 6;
(%o5)                                     12
(%i6) ddf, lnst[2];  /* lnst[2] enth"alt: x = 2 */
(%o6)                                    - 12
(%i7) print ("f(2) = ", f(2), " f(6) = ", f(6))$
f(2) =  32  f(6) =  0
```

Abbildung 5.4: Der Code ist hier: 237

Zuerst wird die Funktion eingegeben.

Dann werden die Ableitungen df und ddf definiert.

Mit **solve** (*[Gleichungen], [Variablen]*) werden die Nullstellen der 1. Ableitung gesucht.

Beim Einsetzen in die 2. Ableitung können Sie entweder die Lösungen der Gleichung von Hand einsetzen: ddf, x = 6; oder Sie können auf die Lösung zugreifen, die Sie in lnst gespeichert haben. ddf, lnst[2]; Dies ist in diesem Fall gleichbedeutend mit ddf, x=2;

print (*Ausdruck, . . .*) druckt Ihnen einen Text. Die übergebenen Ausdrucke werden durch ein Komma getrennt. Zeichenketten, also Wörter bzw. Buchstaben, müssen Sie in Anführungszeichen schreiben.

Nun können Sie die Funktion auch noch zeichnen lassen von -5 bis 5:

```
plot2d([f(x)], [x,-5,5]);
```

5.5 Extremstelle einer Exponentialfunktion

Gesucht ist die Extremstelle von $f(x) = x \cdot e^(2x + 1)$:

Die Funktion können Sie auf drei Arten eingeben. Entweder mit %e oder mit exp(2*x+1). Alternativ können Sie eventuell erst e definieren: e : %e. Dann können Sie auch einfach nur e schreiben.

```
(%i1) e : %e$
(%i2) f(x) := x * e^(2*x + 1);
                                          2 x + 1
(%o2)                          f(x) := x e
(%i3) df : diff( f(x), x);
                                  2 x + 1     2 x + 1
(%o3)                        2 x %e        + %e
(%i4) ddf : diff( f(x), x, 2);
                                 2 x + 1       2 x + 1
(%o4)                       4 x %e        + 4 %e
(%i5) lnst : solve (df = 0, x);
                                   1    2 x + 1
(%o5)                       [x = - -, %e        = 0]
                                   2
(%i6) f : f(x); /* Um f, x=-0.5 schreiben zu koennen */
                                      2 x + 1
(%o6)                                 x %e
(%i7) ddf, first(lnst);
                                        2
(%o7)
(%i8) f, first(lnst);
                                       1
(%o8)                                 - -
                                       2
```

Abbildung 5.5: Der Code ist hier: 238

Beachten Sie, dass Sie in f(x) natürlich von Hand den jeweiligen Wert der Extremstelle eingeben können mit f(-0.5). Aber wenn der Ausdruck länger ist, ist es vielleicht bequemer auf die Liste lnst zugreifen zu können.

Auf das erste Element einer Liste greifen Sie mit **first** (*Liste*) zu. Wenn Sie zwei Nullstellen haben, greifen Sie auf das nächste Listenelement – also auf die zweite Nullstelle – mit **second** (*Liste*) zu. Danach mit **third** (*Liste*) usw.

Durch **numer : true** erhalten Sie die Auswertung als Dezimalzahl.

5.6 Lösen einer Steckbriefaufgabe

Gesucht ist eine Funktion f 4. Grades mit bestimmten Eigenschaften:

f ist 4. Grades	$f(x) = ax^4 + bx^3 + cx^2 + dx + e$
f hat bei 2 einen Sattelpunkt	$f'(2) = 0$
	$f''(2) = 0$
f hat bei $(4\|-128)$ ein Minimum	$f'(4) = 0$
	$f(4) = -128$
Die Fläche zwischen 2 und 4 ist 20	$F(4) - F(2) = 20$

Geben Sie den Funktionswert an der Stelle 1 an.

Lösung

```
(%i1) f(x) := a*x^4 + b*x^3 + c*x^2 + d*x + e;
                                  4      3      2
(%o1)                 f(x) := a x  + b x  + c x  + d x + e
(%i2) df(x) := ''( diff (f(x), x) );
                                    3        2
(%o2)                 df(x) := 4 a x  + 3 b x  + 2 c x + d
(%i3) ddf(x) := ''( diff (f(x), x, 2) );
                                      2
(%o3)                   ddf(x) := 12 a x  + 6 b x + 2 c
(%i4) /* Gleichungen */
gl1 : df(2) = 0$
(%i5) gl2 : ddf(2) = 0$
(%i6) gl3 : df(4) = 0$
(%i7) gl4 : f(4) = -128$
(%i8) gl5 : integrate (f(x), x, 2, 4) = -236.8$
(%i9) lsg : solve ([gl1, gl2, gl3, gl4, gl5], [a, b, c, d, e]);
rat: replaced 236.8 by 1184/5 = 236.8
(%o9)           [[a = 3, b = - 32, c = 120, d = - 192, e = 0]]
(%i10) g(x) :=  ''( subst (lsg, f(x) ) );
                                  4      3        2
(%o10)                g(x) := 3 x  - 32 x  + 120 x  - 192 x
(%i11) g(x);
                                4       3        2
(%o11)                   3 x  - 32 x  + 120 x  - 192 x
```

Abbildung 5.6: Der Code ist hier: 239

Zuerst wird eine Funktion mit den Variablen **a, b, c, d, e** definiert.

Im Anschluss werden die Ableitungen mit **diff (Funktion, Variable, wievielte Ableitung)** gebildet. Dazu wird die Funktion **diff** in Klammern eingeschlossen und durch vorangstellte Hochkommata direkt ausgewertet.

Dann werden die Gleichungen aufgeschrieben und der Übersicht halber benannt, so dass sie hinterher in **solve ([Gleichungen], [Variablen])** aufgerufen und gelöst werden können.

Im letzten Schritt wird die Funktion g definiert. Die Funktion g wird nicht geändert, wenn f oder lsg geändert wird. Die einzelnen Parameter in f werden durch die entsprechenden Werte aus lsg ersetzt mit **subst** (*Ersetzungsliste, Ausdruck*).

5.7 Steckbriefaufgabe erstellen

Sie wollen eine Funktion, die folgende Eigenschaften aufweist:

- $f(x)$ hat bei $x = 2$ eine Wendestelle.
- $f(x)$ hat bei $x = 4$ eine Extremstelle.

Lösung

```
(%i1) g(x) := ''(integrate ((x-2)^2*(x-4), x));
                                 4       3       2
                              3 x  - 32 x  + 120 x  - 192 x
(%o1)                 g(x) := ----------------------------
                                          12
(%i2) f(x) := 12 * g(x);
(%o2)                              f(x) := 12 g(x)
(%i3) f(x) := ''(12 * g(x));
                                 4       3       2
(%o3)                 f(x) := 3 x  - 32 x  + 120 x  - 192 x
(%i4) df(x) := ''(diff (f(x), x, 1));
                                   3       2
(%o4)                 df(x) := 12 x  - 96 x  + 240 x - 192
(%i5) ddf(x) := ''(diff (f(x), x, 2));
                                     2
(%o5)                    ddf(x) := 36 x  - 192 x + 240
(%i6) ddf(4);
(%o6)                                48
(%i7) f(4);
(%o7)                               - 128
(%i8) integrate (f(x), x, 2, 4);
                                     1184
(%o8)                               - ----
                                      5
(%i9) %, numer;
(%o9)                               - 236.8
```

Abbildung 5.7: Der Code ist hier: 240

Sie bestimmen die 1. Ableitung so, dass Sie eine doppelte Nullstelle bei $x = 2$ haben und eine einfache Nullstelle bei $x = 4$. Dieser Ausdruck wird integriert und als Funktion geschrieben.

Um die Funktion zu erhalten klammern Sie den gesamten Ausdruck und schreiben zwei Hochkommata davor. $f(x)$ ist in diesem Fall die mit dem Hauptnenner multiplizierte Funktion.

Dann bestimmen Sie die Ableitungen und können dann einfach feststellen, ob Sie ein Minimum oder Maximum bei $x = 4$ haben.

%, numer gibt Ihnen den letzten Ausdruck als Dezimalzahl.

5.8 Ober- und Untersumme einer monoton steigenden Funktion

Sie wollen die Untersumme und/oder Obersumme einer monoton steigenden Funktion erstellen und mit dem Integral vergleichen. Als Beispiel sei $f(x) = x^2$ und wir bilden die Unter- und Obersumme und das Integral von 2 bis 5.

sum (Ausdruck, Laufvariable, Startwert, Endwert)

```
(%i1) /* Unter- und Obersumme der Funktion x^2 von 2 bis 5  */
f(x) := x^2$
(%i2) n : 100$ /* Anzahl der Schritte */
(%i3) b : (5 - 2) / n$
(%i4) us : sum (b * f(b*i + 2), i, 0, n-1); /* Untersumme */
                                      773709
(%o4)                                 ------
                                      20000
(%i5) os : sum (b * f(b*i + 2), i, 1, n);   /* Obersumme */
                                      786309
(%o5)                                 ------
                                      20000
(%i6) I : integrate (f(x), x, 2, 5); /* Das Integral */
(%o6)                                  39
(%i7) numer : true;              /* Statt Brueche: Dezimalzahlen */
(%o7)                                 true
(%i8) print ("Untersumme: ", us, " Intergral: ", I, " Obersumme: ", os)$
Untersumme:  38.68545  Intergral:  39  Obersumme:  39.31545
```

Abbildung 5.8: Der Code ist hier: 241

Wenn Sie die Anzahl der Schritte leicht variieren wollen, dann bietet es sich an, dass n erst bei der Berechnung der Summe eingesetzt wird:

```
(%i1) f(x) := x^2;
                                        2
(%o1)                            f(x) := x
(%i2) b : (5 - 2) / n;
                                       3
(%o2)                                  -
                                       n
(%i3) us : sum (b * f(b*i + 2), i, 0, n-1);
                                  n - 1
                                  ====
                                  \      3 i     2
                                3  >    (--- + 2)
                                  /       n
                                  ====
                                  i = 0
(%o3)                            ------------------
                                        n
(%i4) os : sum (b * f(b*i + 2), i, 1, n);
                                   n
                                  ====
                                  \      3 i     2
                                3  >    (--- + 2)
                                  /       n
                                  ====
                                  i = 1
(%o4)                            ------------------
                                        n
(%i5) numer : true;
(%o5)                                 true
(%i6) /* Eine einzelne Summe berechnen lassen */
us, n = 100, simpsum;
(%o6)                          38.68545000000001
(%i7)  /* Oder beide Summen gleichzeitig ausdrucken lassen */
print ("Untersumme: ", us, " --- Obersumme = ", os), n = 100, simpsum$
Untersumme:  38.68545000000001  --- Obersumme =  39.31545000000001
(%i8) print ("Untersumme: ", us, " --- Obersumme = ", os), n = 500, simpsum$
Untersumme:  38.937018  --- Obersumme =  39.063018
```

Abbildung 5.9: Der Code ist hier: 242

simpsum : true vereinfacht einen Summenausdruck.

5.9 Erstellen der Ortskurve eines Punktes

Sie haben einen Punkt, dessen x-Koordinate und y-Koordinate von einem Parameter r abhängig ist: P$(2r/8r^2)$

$$x = 2r$$
$$y = 8r^2$$

Sie wollen jetzt eine Ortskurve erstellen. Dazu müssen Sie den y-Wert durch die Variable x ausdrücken (das r entsprechend ersetzen).

$$y = 8r^2 = 2 \cdot 4r^2 = 2 \cdot (2r)^2 = 2x^2$$

```
(%i1) p : [2*r, 8*r^2];
                                          2
(%o1)                              [2 r, 8 r ]
(%i2) solve (x = 2*r, r);
                                        x
(%o2)                             [r = -]
                                        2
(%i3) subst (x/2, r, p);
                                         2
(%o3)                             [x, 2 x ]
(%i4) load (lrats)$
(%i5) k : fullratsubst (p[1] = x, p);
                                         2
(%o5)                             [x, 2 x ]
```

Abbildung 5.10: Der Code ist hier: 243

solve (*Gleichung*, *Variable*) löst die Gleichung. Wenn Sie r als Variable wählen, wird die Gleichung nach „r =“ aufgelöst. **subst** (*a*, *b*, *Term*) ersetzt im Term jedes b durch ein a. Bei uns muss im Term jedes r ersetzt werden durch x/2.

fullratsubst (*Substitutionsanweisung*, *Ausdruck*) liefert ebenfalls das gewünscht Ergebnis. p[1] wird der neue x-Wert und alles andere wird dementsprechend ersetzt.

5.10 Erstellen einer Ortskurve von Extremstellen einer Funktionenschar

Sie suchen die Ortskurve der Extremstellen einer Funktion.

$$f(x) = x^4 - kx^2 \quad k > 0$$

Wenn Sie verschiedene Kurven und die Ortskurve zeichnen wollen, dann passen Sie bitte Kap. 13.8, S. 140 entsprechend an.

```
(%i1) f(x) := x^4 - k*x^2$
(%i2) df : diff (f(x), x);
                                    3
(%o2)                          4 x  - 2 k x
(%i3) ddf : diff (f(x), x, 2);
                                     2
(%o3)                           12 x  - 2 k
(%i4) nst : solve (df = 0, x);
                           sqrt(k)      sqrt(k)
(%o4)              [x = - -------, x = -------, x = 0]
                           sqrt(2)      sqrt(2)
(%i5) n1 : nst[1];
                                      sqrt(k)
(%o5)                          x = - -------
                                      sqrt(2)
(%i6) n2 : nst[2]$
(%i7) n3 : nst[3]$
(%i8) ddf, n1; /* da k > 0 gilt, ist bei n1 ein Minimum */
(%o8)                                 4 k
(%i9) ddf, n2; /* da k > 0 gilt, ist bei n2 ein Minimum */
(%o9)                                 4 k
(%i10) ddf, n3; /* da k > 0 gilt, ist bei n3 ein Maximum */
(%o10)                               - 2 k
(%i11) f(x), n1;
                                       2
                                      k
(%o11)                               - --
                                      4
(%i12) f(x), n2;
                                       2
                                      k
(%o12)                               - --
                                      4
(%i13) f(x), n3;
(%o13)                                0
(%i14) assume (x < 0); /* Wir betrachten die Ortskurve, die sich durch n1
ergibt */
(%o14)                             [x < 0]
(%i15) assume (k > 0);
(%o15)                             [k > 0]
(%i16) ke : solve (n1, k);
                                        2
(%o16)                           [k = 2 x ]
(%i17) ye = f(x), ke; /* f(x), xe */
                                         4
(%o17)                            ye = - x
```

Abbildung 5.11: Der Code ist hier: 244

Legen Sie die Funktion fest und bilden Sie die Ableitungen. Bei den Nullstellen der 1. Ableitung muss noch mit der 2. Ableitung überprüft werden, ob es Extremstellen sind.

Setzen Sie dann die Nullstellen in die Originalfunktion ein. Hier ist die Funktion symmetrisch. Daher reicht es die erste Nullstelle zu betrachten. Die 3. Nullstelle, das Maximum liegt später auf der Ortskurve der Minima. Die erste Nullstelle der ersten Ableitung (also n1) ist bei reellen Zahlen negativ. Die Wurzel von k und von 2 ist jeweils positiv.

assume (*x < 0*) gibt Maxima Hinweise bei der Umformung. Wegen der Wurzelzeichen ist es notwendig sich hier bei dieser speziellen Aufgabe zu entscheiden und Maxima mitzuteilen, dass x kleiner als null ist.

Für kompliziertere Wurzelgleichungen gibt es to_poly_solve (siehe Kap. 4.4, S. 21).

5.11 Taylor Reihe angeben

Sie möchten gerne die Potenzreihenentwicklung der Sinusfunktion bis zur fünften Potenz angeben.

```
(%i1) f(x) := ''( taylor ( sin(x), x, 0, 5 ) );
                                        3    5
                                       x    x
(%o1)/T/                    f(x) := x - -- + --- + . . .
                                       6    120
(%i2) f(%pi / 6);
                              5          3
                             %pi  - 720 %pi  + 155520 %pi
(%o2)/R/                     ----------------------------
                                      933120
(%i3) ''( f(%pi / 6) ), numer;
(%o3)                               .5000021325887924
(%i4) ''( f(%pi / 6) - sin(%pi / 6) ), numer;
(%o4)                            2.1325887924414783E-6
```

Abbildung 5.12: Der Code ist hier: 245

taylor (*Funktion, Variable, Stelle, Grad*) erstellt die Potenzreihe einer Funktion zu einer gegebenen Variablen an einer vorgegebenen Stelle bis zu dem angegebenen Grad. Durch die Klammerung und die zwei vorangestellten Hochkommata erhalten Sie eine Funktion.

Wenn Sie eine Dezimalzahl haben wollen, müssen Sie einen Umweg gehen: Lassen Sie den Ausdruck erst auswerten indem Sie den Ausdruck in eine Klammer einschliessen und zwei Hochkommata voranstellen. Mit einem Komma abgetrennt setzen Sie dann die Optionsvariable numer.

Wenn Sie mit dem Wert aus der Sinusfunktion vergleichen sehen Sie, dass die Näherung gar nicht so schlecht ist.

Kapitel 6

Funktionen

6.1 Eingabe einer Funktion

Sie wollen eine Funktion eingeben.

Das machen Sie mit einem Namen der Funktion (hier: f) und in Klammern schreiben Sie, an welche Variable die übergebenen Werte zugewiesen werden sollen (hier: x).

Statt einen Ausdruck (mit nur einem Doppelpunkt) definieren Sie Funktionen mit einem Doppelpunkt und einem Gleichheitszeichen (:=).

Sie können auf eine Funktion zugreifen, in dem Sie den Wert z. B. die 3 in Klammern schreiben oder indem Sie mit einem Komma abgetrennt angeben, wie gross der x-Wert sein soll.

Die zweite Methode kann insbesondere sinnvoll sein, wenn Sie durch **solve** (*Gleichungen, Variablen*) die Lösungen in der Form x=2 erhalten haben.

```
(%i1) f(x) := 2 * x + 1;
(%o1)                          f(x) := 2 x + 1
(%i2) f(3);
(%o2)                                  7
(%i3) f(x), x = 3;
(%o3)                                  7
```

Abbildung 6.1: Der Code ist hier: 246

Beachten Sie, dass ein Ausdruck und eine Funktion zwei unterschiedliche Dinge sind in Maxima:

```
(%i1) f : x + 1;
(%o1)                                x + 1
(%i2) f(x) := 2*x + 1;
(%o2)                          f(x) := 2 x + 1
(%i3) f, x = 3;
(%o3)                                  4
(%i4) f(3);
(%o4)                                  7
```

Abbildung 6.2: Der Code ist hier: 247

6.2 Funktionen aus der Analysis

Hier sollen kurz die gängigsten Funktionen außer den trigonometrischen Funktionen vorgestellt werden.

```
(%i1) e : %e;
(%o1)                                  %e
(%i2) f(x) := e^(2*x+1);
                                         2 x + 1
(%o2)                          f(x) := e
(%i3) log (2);
(%o3)                               log(2)
(%i4) log (2), numer;
(%o4)                          .6931471805599453
(%i5) sqrt (x^2);
(%o5)                               abs(x)
(%i6) sqrt (x), x = 3;
(%o6)                               sqrt(3)
(%i7) sqrt (x), x = 3, numer;
(%o7)                          1.732050807568877
```

Abbildung 6.3: Der Code ist hier: 248

Weitere Funktionen, die gelegentlich benötigt werden:

```
(%i1) round (0.7);
(%o1)                                  1
(%i2) floor (3.8);
(%o2)                                  3
(%i3) ceiling (0.2);
(%o3)                                  1
(%i4) abs (-4);
(%o4)                                  4
(%i5) abs (4);
(%o5)                                  4
```

Abbildung 6.4: Der Code ist hier: 249

round (x) rundet die Zahl auf oder ab, je nachdem welche der beiden ganzen Zahlen näher ist. 0,6 wird aufgerundet, 0,2 dagegen abgerundet.

floor (x) rundet immer ab.

ceiling (x) rundet dagegen immer auf.

abs (x) gibt den Betrag der Zahl x an. In der Mathematik wird die Zahl mit senkrechten Strichen eingeschlossen. (Der Betrag einer Zahl ist der Abstand der Zahl zum Ursprung.)

$$|-3| = 3$$
$$|3| = 3$$

6.3 Trigonometrie

Sie wollen eine Trigonometrische Funktion benutzen. Ihnen stehen u. a. folgende Funktionen zur Verfügung:
sin (), **cos** (), **tan** ().
Die Hyperbolischen Funktionen:
sinh (), **cosh** (), **tanh** ().
Die Inversen Funktionen erhalen Sie durch ein kleines vorgestelltes a. Also z. B. **asin** ().

```
(%i1) sin (2);
(%o1)                                sin(2)
(%i2) sin (2), numer;
(%o2)                          .9092974268256817
(%i3) sin(x), solve (2*x = 4);
(%o3)                                sin(2)
(%i4) pi : %pi;
(%o4)                                  %pi
(%i5) cos (2 * pi);
(%o5)                                   1
```

Abbildung 6.5: Der Code ist hier: 250

Wenn Sie sin (2) ausgewertet haben wollen, müssen Sie mit Komma abgetrennt ein **numer** ergänzen. Dies ist sinnvoll, denn Sie wollen nicht immer den Wert der Funktion haben. Bei dieser Einstellung in Maxima können Sie entscheiden, ob Sie den Wert ausgewertet haben wollen.

Um nicht immer %pi schreiben zu müssen, empfiehlt es sich eine Variabel namens pi einzuführen, welche eine Abkürzung für %pi ist.

6.4 Funktion ausmultiplizieren

Sie haben eine Funktion z. B. in der Nullstellenform oder eine Parabel in der Scheitelpunktsform und möchten diese Funktion ausmultipliziert haben.

```
(%i1) f(x) := (x-2) * (x+4);
(%o1)                     f(x) := (x - 2) (x + 4)
(%i2) expand ( f(x) );
                                   2
(%o2)                             x  + 2 x - 8
```

Abbildung 6.6: Der Code ist hier: 251

expand (*Ausdruck*) multipliziert den Ausdruck aus und fasst ihn zusammen.

6.5 Wertetabelle erstellen

Sie wollen eine Wertetabelle einer Funktion erstellen.

```
(%i1) g(x) := x^2 + 1;
                                          2
(%o1)                           g(x) := x  + 1
(%i2) /* 1. Moeglichkeit: Ein einzelner Wert */
g(2);
(%o2)                                    5
(%i3) /* 2. Moeglichkeit */
(%i3) xwerte : [-2, -1, 0, 1, 2]; /* x-Werte sind in einer Liste */
(%o3)                          [- 2, - 1, 0, 1, 2]
(%i4) ywerte : map (g, xwerte);   /* y-Werte sind dann auch in einer Liste */
(%o4)                           [5, 2, 1, 2, 5]
(%i5) /* 3. Moeglichkeit als Programm */
(%i5) wertetabelle (term, var, start, ende) := (
(%i5)     print (var, " | ", y ),
(%i5)     for i : start thru ende do
(%i5)         print (i, " | ", subst (i, var, term) )
(%i5)   )$
(%i6) wertetabelle (x^2, x, -2, 2)$
x  |  y
- 2  |  4
- 1  |  1
0  |  0
1  |  1
2  |  4
```

Abbildung 6.7: Der Code ist hier: 252

1. Wenn Sie eine Funktion erstellen, können Sie einzelne Werte ausrechnen lassen.

2. Sie können erst eine Liste mit den x-Werten anlegen und dann eine Liste mit den entsprechenden y-Werten berechnen lassen.

   map (*Funktion, Liste*) erstellt eine neue Liste, bei jedes Element der übergebenen Liste der Funktion übergeben wird. Das erste Argument von **map** ist der Funktionsname g.

3. Sie können aber auch eine Funktion erstellen, welcher Sie den Funktionsausdruck, Startwert und Endwert übergeben.

   Sie umschließen die Anweisungen mit einem block. Da Sie hier keine lokale Variablen deklarieren müssen, reicht es aus den Programmblock in runden Klammern einzuschliessen.

   Darin läuft eine for-Schleife. Die Anweisungen in dem folgenden Block werden mit unterschiedlichen Werten für i durchlaufen.

   subst (*i, var, term*) ersetzt im Term var (welches bei unserem Aufruf ein x darstellt) durch den Wert von i. So werden nacheinander die y-Werte berechnet.

6.6 Ermitteln des Scheitelpunktes einer Parabel

Sie haben eine quadratische Funktion und möchten den Scheitelpunkt ermitteln.

Die Parabel ist symmetrisch um eine Achse durch Scheitelpunkt. Sie benötigen also nur zwei x-Werte, die jeweils denselben y-Wert haben. In der Schule nehmen Sie da oft die Nullstellen. Der x-Wert des Scheitelpunktes ist dann in der Mitte der Nullstellen.

Das Problem ist, dass eine Parabel nicht notwendigerweise zwei Nullstellen hat. Nehmen Sie statt dessen den y-Achsenabschnitt.

```
(%i1) f(x) := 3*x^2 + 12*x + 5;
                                          2
(%o1)                       f(x) := 3 x  + 12 x + 5
(%i2) lx : map (rhs, (solve ( f(x) = f(0) )));
(%o2)                                [- 4, 0]
(%i3) Sx : mean (lx);
(%o3)                            mean([- 4, 0])
(%i4) Sy : f(Sx);
                         2
(%o4)              3 mean ([- 4, 0]) + 12 mean([- 4, 0]) + 5
(%i5) /* Alternative 1 */
Sx : (lx[1] + lx[2]) / 2;
(%o5)                                  - 2
(%i6) Sy : f(Sx);
(%o6)                                  - 7
(%i7) /* Alternative 2 */
SxWert : rhs ((x1 + x2) / 2);
(%o7)                                   0
(%i8) SyWert : f(Sx);
(%o8)                                  - 7
(%i9) print ("Der Scheitelpunkt ist bei (", SxWert, "|", SyWert,").")$
Der Scheitelpunkt ist bei ( 0 | - 7 ).
```

Abbildung 6.8: Der Code ist hier: 253

Definieren Sie die Funktion. Lösen Sie dann die Gleichung: f(x) = f(0) mit **solve** (*Gleichung*). Damit erhalten Sie zwei x-Werte, wenn der Scheitelpunkt nicht auf der y-Achse liegt. **map** (*Funktion*, *Liste*) wendet die Funktion auf jedes Element der übergebenen Liste an. Sie erhalten eine Liste mit genau so vielen Elementen wie Sie die übergebene Liste hat.

rhs (*Gleichung*) gibt Ihnen die rechte Seite der Gleichung.

mean (*Liste*) erstellt Ihnen den Durchschnitt der Liste. lx ist eine Liste aus einem Wert, wenn der Scheitelpunkt auf der y-Achse liegt, oder sonst zwei Werten.

Alternativ können Sie den Durchschnitt auch von Hand berechnen.

Wenn Sie mit Werten arbeiten wollen, dann benötigen Sie in der Alternative 2 **rhs** (*Gleichung*).

Mit **print** (*Ausdruck*, ...) können Sie hier einen Satz ausgeben. Zeichenketten werden in Anführungszeichen gesetzt und Variablen, deren Wert ausgegeben werden soll werden ohne Anführungszeichen gesetzt. Die einzelnen Ausdrücke werden mit einem Komma abgetrennt.

6.7 Maximaler Funktionswert in einem Intervall

Sie haben eine stetige Funktion f und ein Intervall: [a,b].

Sie möchten nun den größten Funktionswert der Funktion f zwischen 0 und 3 erhalten.

$$f(x) = 3x^4 - 20x^3 + 12x^2 + 96$$

```
(%i1) f(x) := 3 * x^4  - 20 * x^3  + 12 * x^2  + 96 * x$
(%i2) /* Intervallgrenzen */
a : 0$
(%i3) b : 3$
(%i4)
df : diff (f(x), x)$
(%i5) nst_lsg_ableitung : solve (df = 0, x);
(%o5)                          [x = 4, x = - 1, x = 2]
(%i6) x_nst_lsg_ableitung : map (rhs, (nst_lsg_ableitung));
(%o6)                                [4, - 1, 2]
(%i7) xpos : append ([a], [b], x_nst_lsg_ableitung);
(%o7)                            [0, 3, 4, - 1, 2]
(%i8) xpos_Intervall_ind : sublist_indices (xpos,
                lambda ([x], x >= a and x <= b) );
(%o8)                                [1, 2, 5]
(%i9) xwerte_Intervall : makelist (xpos[x], x, xpos_Intervall_ind);
(%o9)                                [0, 3, 2]
(%i10) ywerte : create_list (f(x), x, xwerte_Intervall);
(%o10)                             [0, 99, 128]
(%i11)
lmax (ywerte);
(%o11)                                 128
```

Abbildung 6.9: Der Code ist hier: 254

Wir nutzen aus, dass eine Funktion ihre größten Werte jeweils am Rand oder an den Extremstellen annimmt. Wir suchen hier alle x-Werte, bei denen die 1. Ableitung null wird. Dies können auch Sattelpunkte sein. Aber da der Vergleich automatisch erfolgt, reicht einfaches Einsetzen in die Funktion f.

Zuerst wird die Funktion definiert. Anschliessend werden die Intervallgrenzen festgelegt. a ist dabei der untere Wert des Intervalls und b der obere.

diff (*Funktion*, *Variable*) leitet die Funktion f(x) mit der Variablen x ab.

solve (*Gleichung*, *Variable*) berechnet die Nullstellen der 1. Ableitung. Dies ergibt eine Liste mit den Elementen:

- **1. Element:** x = 4
- **2. Element:** x = -1
- **3. Element:** x = 2

Jedes Element ist eine Gleichung.

In der folgenden Anweisung werden die einzelnen x-Werte in einer Liste erstellt. Da die gesuchten x-Werte alle rechts von dem Gleichheitszeichen stehen, gibt **rhs** (*Ausdruck*) gerade die x-Werte. **rhs** (*Ausdruck*) gibt die „rechte“ Seite eines Ausdrucks zurück. Der Ausdruck wird getrennt durch =, $<$, $\leq$, equal usw.

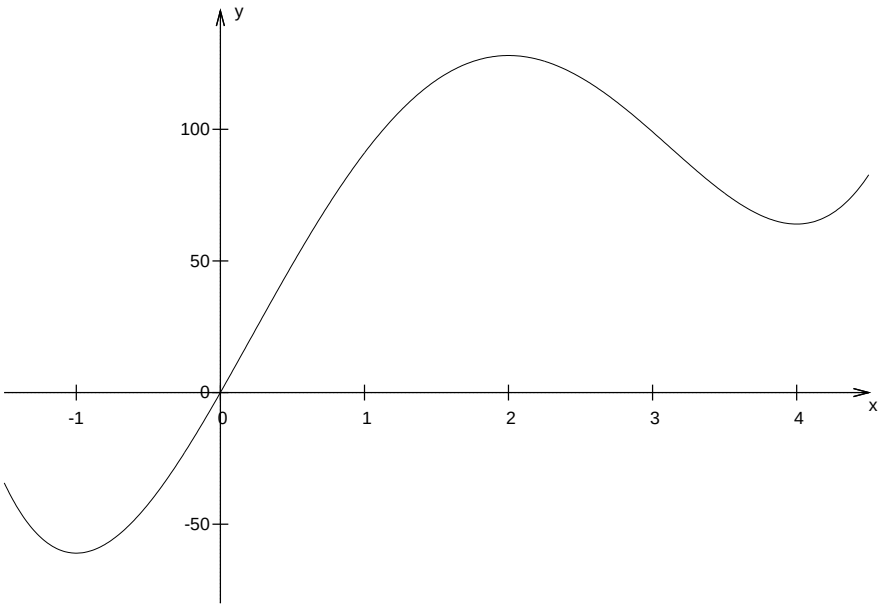


Abbildung 6.10: $f(x) := 3x^4 - 20x^3 + 12x^2 + 96x$
Die Funktion hat ihre Extremstellen bei -1, 2 und 4.

Nun werden alle x-Werte der gefundenen Nullstellen der 1. Ableitung und die x-Werte der Ränder in einer Liste zusammengefasst mit **append** (*Liste1, List2, . . .*).

sublist_indices (*Liste, Funktion*) gibt die Positionen (Indices) der Liste, bei der die Funktion ein „true“ zurückliefert. Hier wird überprüft, welche x-Werte größer als a und kleiner als b sind mit Hilfe der anonymen Lambda-Funktion. (Siehe auch: Kapitel: 12.7, S. 125).

Bei **lambda** (*[lokale Var], Anweisungen*) werden die einzelnen Werte der Liste an x übergeben. Dann wird getestet ob x größer als a und kleiner als b ist und entsprechend ein „wahr“ oder „falsch“ zurückgegeben. Bei einem „wahr“ wird die Position in der Liste vermerkt.

Mit **makelist** (*Ausdruck, Variable, Liste*) wird jetzt die Liste erstellt, die die x-werte innerhalb des Intervalls enthält (inklusive der Ränder). makelist erstellt eine Liste, welches gleichviele Elemente wie Liste (3. Argument von makelist) enthält.

Die Variable x in xpos[x] wird nacheinander durch die Elemente von xpos_Intervall_ind ersetzt.

Mit **create_list** (*Ausdruck, Variable, Liste*) wird nun eine Liste der dazugehörigen y-Werte erstellt. x durchläuft die x-Werte der Liste xwerte_Intervall in f(x). Es wird also die Liste [f(0), f(3), f(2)] erstellt.

lmax (*Liste*) bestimmt das größte Element einer Liste.

Zum Vergleich: **max** (*zahl1, zahl2, . . .*) bestimmt die größte der übergebenen Zahlen.

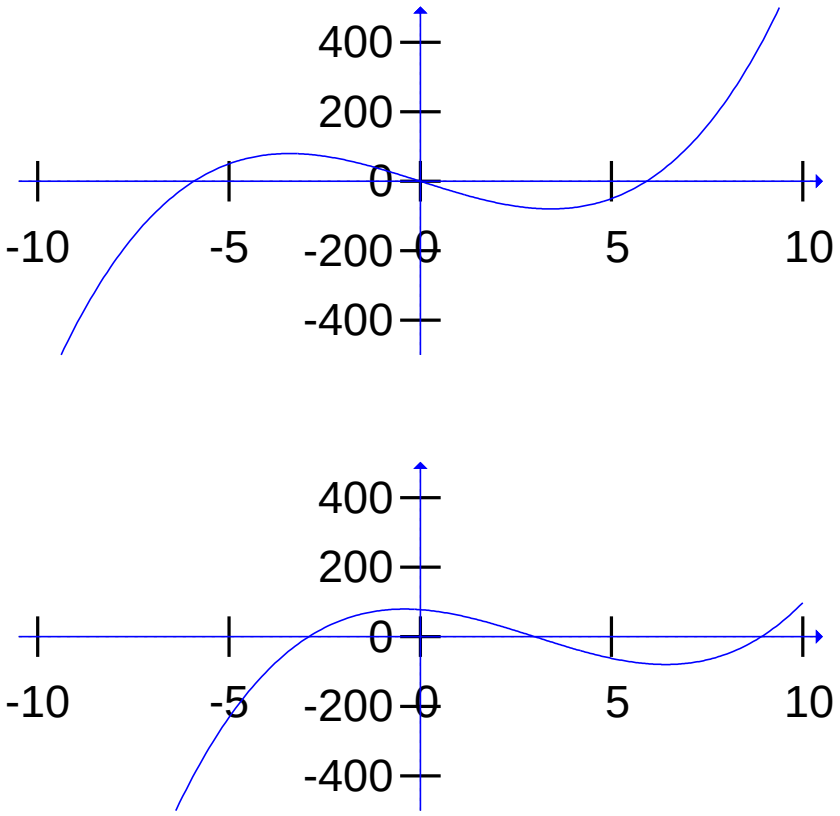
6.8 Den Graph einer Funktion verschieben

Sie haben eine Funktion f und wollen den Graphen um 3 nach rechts verschieben:

$$f(x) := x^3 + 2x - 5$$

```
(%i1) load (draw)$
(%i2) f(x) := x^3 - 30*x -5*x$
(%i3) g : subst ( x-3, x, f(x) );
                                    3
(%o3)                        (x - 3)  - 35 (x - 3)
(%i4)
set_draw_defaults (
  xrange = [-10.5, 10.5],
  yrange = [-500, 500],
  user_preamble = ["set noborder;"],
  xtics_axis = true,
  ytics_axis = true,
  xaxis = true,
  yaxis = true,
  head_length = 0.2,
  dimensions = [500, 500],
  file_name = "verschobeneFunktion",
  terminal = jpg
)$
(%i5)
koordinaten : [
    vector ([-10.5, 0], [21, 0]),
    vector ([0, -500], [0, 1000])
  ]$
(%i6)
draw (
  gr2d (
    explicit ( f(x), x, -10, 10),
    koordinaten
  ),
  gr2d (
    explicit ( g, x, -10, 10),
    koordinaten
  )
)$
```

Abbildung 6.11: Der Code ist hier: 255



load (*draw*) lädt das Zeichenpaket.

Zuerst definieren Sie die Funktion f(x).

Mit **subst** (*ersetzendes, zu ersetzendes, Ausdruck*) wird jedes x durch (x-3) in f(x) ersetzt. Beachten Sie, dass g in Maxima jetzt keine Funktion ist, sondern „nur“ ein Ausdruck. Wenn Sie g in eine Funktion umwandeln wollen, müssen Sie folgende Zeile einfügen:
g(x) := ev (g, ’x = x);

In set_draw_defaults werden die Standardwerte für die beiden Bilder beschrieben. **user_preamble** sorgt dafür, dass kein Rahmen um das Bild gezeichnet wird. **xtics_axis** zeichnet die xtics auf die x-Achse. **head_length** passt die Pfeilgrößen der Vektoren an. **dimensions** passt die Größe des Bildes an.

In koordinaten werden die Pfeile gezeichnet, welche das Koordinatensystem ausmachen. **vector**([x, y], [Δx, Δy] zeichnet ausgehend von einem Startpunkt bei (x, y) dann den Pfeil um Δx in x-Richtung und um Δy in y-Richtung.

draw zeichnet dann die Bilder.

Kapitel 7

Matrizen

7.1 Eingabe von Matrizen

7.1.1 Als Listen

Bei Matrizen werden die Zeilen als Listen eingegeben:

```
(%i1) M : matrix(
    [1,2],
    [3,4]
);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) a : [1, 2];
(%o2)                                [1, 2]
(%i3) b : [3, 4];
(%o3)                                [3, 4]
(%i4) B : matrix (a, b);
                                    [ 1  2 ]
(%o4)                               [      ]
                                    [ 3  4 ]
```

Abbildung 7.1: Der Code ist hier: 256

Es gibt noch sehr viele weitere Möglichkeiten eine Matrix zu füllen (und teilweise auch dargestellt). Diese sind in der Regel in der Schule jedoch von geringem Interesse.

7.1.2 Vektoren

Vektoren bzw. 1-spaltige Matrizen lassen sich auch mit **columnvector** (*Liste*) oder **covect** (*Liste*) eingeben. Dazu muss vorher das Paket „eigen“ geladen werden.

```
(%i1) load (eigen)$
(%i2) v : columnvector ([1, 2, 3]);
                                    [ 1 ]
                                    [   ]
(%o2)                               [ 2 ]
                                    [   ]
                                    [ 3 ]
(%i3) v2 : covect ([1, 2, 3]);
                                    [ 1 ]
                                    [   ]
(%o3)                               [ 2 ]
                                    [   ]
                                    [ 3 ]
```

Abbildung 7.2: Der Code ist hier: 257

7.1.3 Kopieren von Matrizen

Eine Matrix kann zwei verschiedene Namen haben. Wenn Sie auch zwei verschiedene Matrizen haben wollen, müssen Sie **copymatrix** (*Matrix*) benutzten:

```
(%i1) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) B : M;  /* B ist ein Verweis auf M */
                                    [ 1  2 ]
(%o2)                               [      ]
                                    [ 3  4 ]
(%i3) M[1][2] : 5; /* Bei M wird M(1,2) auf 5 gesetzt */
(%o3)                                  5
(%i4) M;
                                    [ 1  5 ]
(%o4)                               [      ]
                                    [ 3  4 ]
(%i5) B;      /* Hier ist ebenfalls aus der 2 eine 5 geworden */
                                    [ 1  5 ]
(%o5)                               [      ]
                                    [ 3  4 ]
(%i6) /* Aber */
(%i6) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o6)                               [      ]
                                    [ 3  4 ]
(%i7) C : copymatrix (M);
                                    [ 1  2 ]
(%o7)                               [      ]
                                    [ 3  4 ]
(%i8) M[1][2] : 5;
(%o8)                                  5
(%i9) M;
                                    [ 1  5 ]
(%o9)                               [      ]
                                    [ 3  4 ]
(%i10) C;      /* Weil C ein neues Objekt ist, ist C nicht veraendert */
                                    [ 1  2 ]
(%o10)                              [      ]
                                    [ 3  4 ]
```

Abbildung 7.3: Der Code ist hier: 258

Im ersten Fall gibt es eine Matrix mit zwei Namen (M und B) und im zweiten Fall zwei Matrizen (M und C) mit jeweils unterschiedlichem Namen. Im ersten Fall betrifft eine Änderung an M auch B, aber im zweiten Fall haben Sie zwei unterschiedliche Matrizen. Eine Änderung an M lässt C unberührt.

7.1.4 Eingabe spezieller Matrizen

7.1.5 Die Einheitsmatrix

Die Einheitsmatrix:

```
(%i1) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) print (ident (2), identfor(M))$
0 errors, 0 warnings
[ 1  0 ] [ 1  0 ]
[      ] [      ]
[ 0  1 ] [ 0  1 ]
```

Abbildung 7.4: Der Code ist hier: 259

Wenn Sie schon eine Matrix haben und dazu eine Einheitsmatrix benötigen, z. B. um den Fixpunkt, bzw. die stationäre Lösung zu bestimmen $(M\vec{x} = \vec{x})$, dann können Sie auch **identfor** (*Matrix*) verwenden:

```
(%i1) M : matrix ([1, 2, 3], [2, 3, 4]);
                                    [ 1  2  3 ]
(%o1)                               [         ]
                                    [ 2  3  4 ]
(%i2) B : matrix ([1, 2, 3], [2, 3, 4], [3, 4, 5]);
                                    [ 1  2  3 ]
                                    [         ]
(%o2)                               [ 2  3  4 ]
                                    [         ]
                                    [ 3  4  5 ]
(%i3) print (ident (2), identfor(M), identfor(B))$
0 errors, 0 warnings
                      [ 1  0  0 ]
[ 1  0 ] [ 0  1  0 ] [         ]
[      ] [         ] [ 0  1  0 ]
[ 0  1 ] [ 0  0  1 ] [         ]
                      [ 0  0  1 ]
```

Abbildung 7.5: Der Code ist hier: 260

Dabei sollten Sie aber sicher stellen, um keine unangenehmen Überraschungen zu erleiden, dass Sie eine quadratische Matrix haben.

7.1.6 Eine Matrix mit wenigen von Null verschiedenen Zahlen

Wenn Sie eine Matrix mit vielen Nullen eingeben wollen, dann können Sie erst eine Matrix mit Nullen erzeugen und dann einzelne Werte von Hand gezielt abändern:

```
(%i1) M : zeromatrix (2, 3);
                                    [ 0  0  0 ]
(%o1)                               [         ]
                                    [ 0  0  0 ]
(%i2) M[1][2] : 3;
(%o2)                                  3
(%i3) M;
                                    [ 0  3  0 ]
(%o3)                               [         ]
                                    [ 0  0  0 ]
```

Abbildung 7.6: Der Code ist hier: 261

7.1.7 Erstellen einer Matrix mit einer Zahl

Sie wollen eine Matrix erstellen, deren Elemente alle dieselbe Zahl sind.

Erstellen Sie eine Matrix mit Nullen und addieren Sie dann eine Zahl.

```
(%i1) M : zeromatrix (2, 2);
                                    [ 0  0 ]
(%o1)                               [      ]
                                    [ 0  0 ]
(%i2) M + 1;
                                    [ 1  1 ]
(%o2)                               [      ]
                                    [ 1  1 ]
```

Abbildung 7.7: Der Code ist hier: 262

7.1.8 Diagonalmatrix erzeugen

Wenn Sie nur eine Diagonalmatrix mit demselben Wert wollen:

```
(%i1) diagmatrix (2, 3);
                                    [ 3  0 ]
(%o1)                               [      ]
                                    [ 0  3 ]
```

Abbildung 7.8: Der Code ist hier: 263

7.1.9 Füllen einer Matrix durch eine Funktion

Sie können die Matrix auch durch eine Funktion füllen. Hier wird die Matrix mit einer Zahl gefüllt:

```
(%i1) h [i, j] := 1;
                                        h     := 1
(%o1)                                    i, j
(%i2) genmatrix (h, 2, 2);
                                        [ 1  1 ]
(%o2)                                   [      ]
                                        [ 1  1 ]
```

Abbildung 7.9: Der Code ist hier: 264

h ist eine Funktion, die von i (Zeile) und j (Spalte) unabhängig immer dieselbe Zahl nämlich die 1 liefert. Vergleichen Sie dieses Beispiel auch mit 7.1.7.

Sie können auch eine anonyme Funktion **lambda** (*[i, j], Anweisungen*) deklarieren. lambda ist hier von den Variablen i, j abhängig und gibt unabhängig von i und j immer 1 zurück. i und j sind Zeile und Spalte.

```
(%i1) genmatrix (lambda ([i,j], 1), 2, 2);
                                        [ 1  1 ]
(%o1)                                   [      ]
                                        [ 1  1 ]
```

Abbildung 7.10: Der Code ist hier: 265

Oder mit einer etwas komplizierteren Funktion.

```
(%i1) M : genmatrix (lambda ([i, j], 3*(i-1) + j), 3, 3);
                                       [ 1  2  3 ]
                                       [         ]
(%o1)                                  [ 4  5  6 ]
                                       [         ]
                                       [ 7  8  9 ]
```

Abbildung 7.11: Der Code ist hier: 266

genmatrix (*..., 3, 3*) erzeugt eine (3×3)-Matrix. Die einzelnen Elemente werden durch die anonyme Funktion lambda erstellt.

lambda (*[i, j], 3*(i-1) + j*) ist eine Funktion, welche von den beiden Variablen i und j abhängig ist. Der Rückgabewert der Funktion ist gerade: 3*(i-1) + j.

7.2 Eigenschaften einer Matrix

7.2.1 Bestimmen der Spur einer Matrix

Wenn Sie die Spur einer Matrix (also die Summe der Elemente der Hauptdiagonalen) ermitteln wollen, benutzen Sie **mat_trace** (*M*).

Alternativ können Sie das Paket „nchrpl“ laden mit **load** (*nchrpl*) und dann die Funktion **mattrace** (*M*) benutzen. Die Funktionsnamen unterscheiden sich nur durch den Unterstrich.

```
(%i1) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) spur : mat_trace (M);
0 errors, 0 warnings
(%o2)                                  5
(%i3) /* Alternative */
load (nchrpl)$
(%i4) spur : mattrace (M);
(%o4)                                  5
```

Abbildung 7.12: Der Code ist hier: 267

7.2.2 Bestimmen der Determinante einer Matrix

```
(%i1) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) determinant (M);
(%o2)                                 - 2
```

Abbildung 7.13: Der Code ist hier: 268

Die Determinante einer Matrix erhalten Sie durch **determinant** (*Matrix*).

mat_trace (*Matrix*) gibt die Spur einer Matrix zurück. Wenn Sie das Paket **load** (*nchrpl*) laden, können Sie auch **mattrace** (*Matrix*) verwenden.

7.2.3 Bestimmen des Ranges einer Matrix

```
(%i1) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) rank (M);
(%o2)                                  2
```

Abbildung 7.14: Der Code ist hier: 269

Den Rang einer Matrix erhalten Sie durch **rank** (*Matrix*). Der Rang einer Matrix ist die Anzahl der von Null verschiedenen Zeilen am Ende des Gauß-Algorithmus.

7.2.4 Angabe der Reihen- und Spaltenanzahl einer Matrix

```
(%i1) M : matrix ([1, 2, 3], [4, 5, 6]);
                                    [ 1  2  3 ]
(%o1)                               [         ]
                                    [ 4  5  6 ]
(%i2) matrix_size (M);
0 errors, 0 warnings
(%o2)                                 [2, 3]
(%i3) reihenanzahl : matrix_size (M)[1];
(%o3)                                    2
(%i4) spaltenanzahl : matrix_size (M)[2];
(%o4)                                    3
```

Abbildung 7.15: Der Code ist hier: 270

matrix_size (*M*) gibt Ihnen als eine Liste mit zwei Elementen die Reihenanzahl und Spaltenanzahl zurück.

7.3 Matrizen Operationen

In diesem Abschnitt soll kurz vorgestellt werden, wie Sie mit Hilfe von Maxima mit Matrizen rechnen.

7.3.1 Matrizen addieren

Matrizen werden komponentenweise addiert. Sie könenn auch alle Werte um eine Zahl vergrößern:

```
(%i1) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) B : matrix ([1, 2], [1, 2]);
                                    [ 1  2 ]
(%o2)                               [      ]
                                    [ 1  2 ]
(%i3) M + B;
                                    [ 2  4 ]
(%o3)                               [      ]
                                    [ 4  6 ]
(%i4) M + 2;
                                    [ 3  4 ]
(%o4)                               [      ]
                                    [ 5  6 ]
```

Abbildung 7.16: Der Code ist hier: 271

7.3.2 Matrizen multiplizieren

Beim Multiplizieren müssen Sie aufpassen. **Die normale Matrixmultiplikation erfolgt mit dem Punkt: „ . “.** Der Stern: „*“ bedeutet, dass elementweise multipliziert wird. Bei einem Stern werden bei M*M die Elemente folgendermaßen berechnet:
$m_{11} = m_{11} \cdot m_{11}$
$m_{12} = m_{12} \cdot m_{12}$, usw.

Sie können natürlich auch jedes Matrixelement mit einer Zahl multiplizieren.

```
(%i1) M : matrix ([1, 2], [0, 2]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 0  2 ]
(%i2) B : diagmatrix (2, 3);
                                    [ 3  0 ]
(%o2)                               [      ]
                                    [ 0  3 ]
(%i3) B * M; /* Vorsicht: elementweise Multiplikation */
                                    [ 3  0 ]
(%o3)                               [      ]
                                    [ 0  6 ]
(%i4) B . M; /* Matrizenmultiplikation */
                                    [ 3  6 ]
(%o4)                               [      ]
                                    [ 0  6 ]
(%i5) M . 2;
                                    [ 2  4 ]
(%o5)                               [      ]
                                    [ 0  4 ]
(%i6) M * 2; /* Bei einer Zahl ist es egal ob . oder * */
                                    [ 2  4 ]
(%o6)                               [      ]
                                    [ 0  4 ]
```

Abbildung 7.17: Der Code ist hier: 272

7.3.3 Matrizen potenzieren

Beim Matrizenmultiplizieren müssen Sie ebenfalls aufpassen: Bei einem „^“ werden die einzelnen Elemente der Matrix potenziert. **Erst bei zwei „^^“ wird tatsächlich M^2 ausgerechnet:**

```
(%i1) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) M^2;  /* Elementweises Potenzieren. nicht M hoch 2 */
                                    [ 1  4  ]
(%o2)                               [       ]
                                    [ 9  16 ]
(%i3) M^^2; /* = M . M */
                                   [ 7   10 ]
(%o3)                              [        ]
                                   [ 15  22 ]
(%i4) M*M;  /* Elementweises Multiplizieren */
                                    [ 1  4  ]
(%o4)                               [       ]
                                    [ 9  16 ]
(%i5) M.M;  /* Matrixmultiplikation */
                                   [ 7   10 ]
(%o5)                              [        ]
                                   [ 15  22 ]
```

Abbildung 7.18: Der Code ist hier: 273

Beachten Sie auch hier wieder den Unterschied zwischen M.M und M*M.

7.3.4 Matrizen invertieren

Es gibt zwei Möglichkeiten. Entweder Sie lassen ^^-1 rechnen oder Sie benutzen **invert** (*Matrix*).

```
(%i1) M : matrix (
    [1, 2],
    [3, 4]
);
                                  [ 1  2 ]
(%o1)                             [      ]
                                  [ 3  4 ]
(%i2) C : M^^-1;
                                 [ - 2   1  ]
                                 [          ]
(%o2)                            [  3     1 ]
                                 [  -   - - ]
                                 [  2     2 ]
(%i3) D : invert (M);
                                 [ - 2   1  ]
                                 [          ]
(%o3)                            [  3     1 ]
                                 [  -   - - ]
                                 [  2     2 ]
(%i4) M . C;
                                  [ 1  0 ]
(%o4)                             [      ]
                                  [ 0  1 ]
(%i5) M . D;
                                  [ 1  0 ]
(%o5)                             [      ]
                                  [ 0  1 ]
```

Abbildung 7.19: Der Code ist hier: 274

7.3.5 Zerteilen einer Matrix

Sie wollen aus einer erweiterten Matrix die Matrix bzw. den Lösungsvektor herauslösen. Dies bewerkstelligen Sie mit **submatrix** ($i_1, \ldots i_m, M, j_1, \ldots, j_n$). **submatrix** (. . .) löscht die angegebenen Zeilen $(i_1, \ldots i_m)$ und Spalten $(j_1, \ldots, j_n)$ der Matrix M. Vor der Matrix stehen die zu löschenden Zeilen und hinter der Matrix die zu löschenden Spalten.

```
(%i1) M : matrix (
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
    );
                                  [ 1  2  3 ]
                                  [         ]
(%o1)                             [ 4  5  6 ]
                                  [         ]
                                  [ 7  8  9 ]
(%i2) submatrix (M, 3);
                                   [ 1  2 ]
                                   [      ]
(%o2)                              [ 4  5 ]
                                   [      ]
                                   [ 7  8 ]
(%i3) letzteSpalte : submatrix (M, 1, 2);
                                     [ 3 ]
                                     [   ]
(%o3)                                [ 6 ]
                                     [   ]
                                     [ 9 ]
(%i4) ersteZeile : submatrix (2, 3, M);
(%o4)                             [ 1  2  3 ]
```

Abbildung 7.20: Der Code ist hier: 275

Wenn Sie nur eine einzelne Zeile oder Spalte erhalten wollen, dann geht das auch mit **col** (*M*, *i*) und **row** (*M*, *j*).

```
(%i1) M : matrix (
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
    );
                                  [ 1  2  3 ]
                                  [         ]
(%o1)                             [ 4  5  6 ]
                                  [         ]
                                  [ 7  8  9 ]
(%i2) col (M, 3);
                                     [ 3 ]
                                     [   ]
(%o2)                                [ 6 ]
                                     [   ]
                                     [ 9 ]
(%i3) row (M, 3);
(%o3)                             [ 7  8  9 ]
```

Abbildung 7.21: Der Code ist hier: 276

7.3.6 Zusammenfügen von Matrizen

Sie wollen eine erweiterte Matrix erstellen. Dazu müssen Sie die Matrix und den Lösungsvektor zusammenfügen. Dies erfolgt durch **addcol** (*M*, *list1*, . . . , *listn*)

```
(%i1) load (eigen);
(%o1)         /usr/local/share/maxima/5.26.0/share/matrix/eigen.mac
(%i2) M : matrix (
          [1, 2],
          [3, 4]
        );
                                      [ 1  2 ]
(%o2)                                 [      ]
                                      [ 3  4 ]
(%i3) v : covect ( [5, 6] );
                                        [ 5 ]
(%o3)                                   [   ]
                                        [ 6 ]
(%i4) addcol (M, v);
                                     [ 1  2  5 ]
(%o4)                                [         ]
                                     [ 3  4  6 ]
(%i5) addcol (M, v, v);
                                  [ 1  2  5  5 ]
(%o5)                             [            ]
                                  [ 3  4  6  6 ]
(%i6) addcol (M, M);
                                  [ 1  2  1  2 ]
(%o6)                             [            ]
                                  [ 3  4  3  4 ]
```

Abbildung 7.22: Der Code ist hier: 277

Sie können auch mehrere Listen übergeben und direkt anhängen.

In derselben Weise kann Maxima Reihen und Matrizen aneinanderfügen. Dazu benötigen Sie **addrow** (*M*, *list1*, *list2*, . . .).

```
(%i1) M : matrix (
        [1, 2],
        [3, 4]
    );
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 3  4 ]
(%i2) v : [5, 6];
(%o2)                                [5, 6]
(%i3) addrow (M, v);
                                    [ 1  2 ]
                                    [      ]
(%o3)                               [ 3  4 ]
                                    [      ]
                                    [ 5  6 ]
```

Abbildung 7.23: Der Code ist hier: 278

7.4 Gleichungen lösen mit Matrizen

7.4.1 Erstellen einer erweiterten Matrix

Eine Matrix kann mit einem Vektor kombiniert werden mit Hilfe von **addcol** (*Matrix*). Der Vollständigkeit halber sei hier auch **addrow** (*Matrix*) erwähnt.

```
(%i1) load(eigen);
(%o1)           /usr/share/maxima/5.22.1/share/matrix/eigen.mac
(%i2) M : matrix ([1, 2], [3, 4]);
                                    [ 1  2 ]
(%o2)                               [      ]
                                    [ 3  4 ]
(%i3) B : columnvector ([5, 6]);
                                      [ 5 ]
(%o3)                                 [   ]
                                      [ 6 ]
(%i4) C : addcol (M, B);
                                   [ 1  2  5 ]
(%o4)                              [         ]
                                   [ 3  4  6 ]
(%i5) D : addrow (M, B);
                                    [ 1  2 ]
                                    [      ]
(%o5)                               [ 3  4 ]
                                    [      ]
                                    [ 5  6 ]
```

Abbildung 7.24: Der Code ist hier: 279

7.4.2 Lösung der Gleichung Mx=0

Um die Gleichung $M\vec{x} = 0$ zu lösen, können Sie die Funktion **nullspace** (*Matrix*) verwenden. Diese Gleichung erhalten Sie, wenn Sie z. B. den Fixpunkt bzw. die stationäre Lösung berechnen wollen. **nullspace** (*M*) einer Matrix M berechnet die Lösungen der Gleichung: $M\vec{x} = 0$.

span gibt an, dass alle Vielfachen von $\binom{-2}{1}$ eine Lösung sind. Das Problem ist, dass Sie auf diese Vektoren nur mit Tricks zugreifen können.

```
(%i1) M : matrix (
    [1, 2],
    [1, 2]
  );
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 1  2 ]
(%i2) c : nullspace (M);
0 errors, 0 warnings
                                       [ - 2 ]
(%o2)                             span([     ])
                                       [  1  ]
(%i3) first (c);
                                     [ - 2 ]
(%o3)                                [     ]
                                     [  1  ]
```

Abbildung 7.25: Der Code ist hier: 280

Wenn Sie also mit dem Vektor weiterrechnen wollen, geht das mit **first**, bei mehreren Vektoren auch mit **second** etc. Sie können alternativ auch **args(c)** verwenden. Das ist besonders praktisch, wenn Sie eine Schleife benutzen wollen.

```
(%i1) B : matrix ([1, 2, 0], [0, 0, 0], [0, 0, 0]);
                                  [ 1  2  0 ]
                                  [         ]
(%o1)                             [ 0  0  0 ]
                                  [         ]
                                  [ 0  0  0 ]
(%i2) c : nullspace (B);
0 errors, 0 warnings
                                  [ - 2 ]  [  0  ]
                                  [     ]  [     ]
(%o2)                        span([  1  ], [  0  ])
                                  [     ]  [     ]
                                  [  0  ]  [ - 2 ]
(%i3) first (c);
                                    [ - 2 ]
                                    [     ]
(%o3)                               [  1  ]
                                    [     ]
                                    [  0  ]
(%i4) args (c);
                                 [ - 2 ]  [  0  ]
                                 [     ]  [     ]
(%o4)                           [[  1  ], [  0  ]]
                                 [     ]  [     ]
                                 [  0  ]  [ - 2 ]
(%i5) args (c)[1];
                                    [ - 2 ]
                                    [     ]
(%o5)                               [  1  ]
                                    [     ]
                                    [  0  ]
```

Abbildung 7.26: Der Code ist hier: 281

args (*c*) gibt Ihnen eine Liste von Vektoren zurück. Auf die einzelnen Vektoren können Sie dann mit **args** (*c*)[1] bzw. **args** (*c*)[2] zugreifen.

7.4.3 Gleichungen mit einer Lösung

Wenn es nur genau eine Lösung gibt (also die Determinante der Matrix ungleich null ist), dann ist die Matrix M auch invertierbar:

$$
\begin{aligned}
M\,\vec{x} &= \vec{v} && \text{Multiplikation von links mit der Inversen}\\
M^{-1}\,M\,\vec{x} &= M^{-1}\,\vec{v}\\
\vec{x} &= M^{-1}\,\vec{v}
\end{aligned}
$$

```
(%i1) M : matrix( [1, 2], [2, 1] );
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 2  1 ]
(%i2) determinant(M);
(%o2)                                - 3
(%i3) v : matrix( [5], [7] );
                                      [ 5 ]
(%o3)                                 [   ]
                                      [ 7 ]
(%i4) x : M^^-1 . v;
                                      [ 3 ]
(%o4)                                 [   ]
                                      [ 1 ]
(%i5) x : invert (M) . v;
                                      [ 3 ]
(%o5)                                 [   ]
                                      [ 1 ]
```

Abbildung 7.27: Der Code ist hier: 282

Mit **determinant** (*M*) ermitteln Sie die Determinante der Matrix M. Wenn die Determinante ungleich null ist, dann können Sie die Inverse der Matrix bilden. Statt: M^^-1 können Sie auch die Inverse mit der Funktion **invert** (*M*) bestimmen lassen.

7.4.4 Gleichungen mit mehreren Lösungen

Maxima kann eigentlich nicht diese Gleichungen in Matrizenform lösen. Dennoch gibt es einige Tricks, die Gleichungen trotzdem adäquat zu lösen. Dazu müssen die Matrizen wieder in Gleichungen zurückgeführt werden und können dann anschließend mit **solve** (*Gleichungenliste, Variablenliste*) gelöst werden.

In Gleichungen umformen

Wenn Sie mehrere Lösungen haben, also die Determinante der Matrix null ist, dann können Sie die Matrix nicht invertieren. Sie können die Gleichungen erstellen und dann mit **solve** lösen.

Hier wird die Gleichung $M\vec{x} = \vec{v}$ gelöst.

```
(%i1) M : matrix( [1, 2],[1, 2] );
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 1  2 ]
(%i2) v : matrix( [5], [5] );
                                      [ 5 ]
(%o2)                                 [   ]
                                      [ 5 ]
(%i3) x : matrix( [x1], [x2] );
                                     [ x1 ]
(%o3)                                [    ]
                                     [ x2 ]
(%i4) C : M.x;
                                  [ 2 x2 + x1 ]
(%o4)                             [           ]
                                  [ 2 x2 + x1 ]
(%i5) gl1 : C[1, 1] = v[1, 1];
(%o5)                             2 x2 + x1 = 5
(%i6) gl2 : C[2, 1] = v[2, 1];
(%o6)                             2 x2 + x1 = 5
(%i7) solve( [gl1, gl2], [x1, x2] );
solve: dependent equations eliminated: (2)
(%o7)                  [[x1 = 5 - 2 %r1, x2 = %r1]]
```

Abbildung 7.28: Der Code ist hier: 283

Sie multiplizieren die Matrix M mit dem Vektor x, so dass Sie einen Vektor erhalten, in dem der linke Teil der Gleichung mit den Variablen x1 und x2 geschrieben ist. Dann werden die Gleichungen gelöst.

$$\begin{pmatrix} x1 \\ x2 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} + r \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

In Gleichungen durch eine Funktion umformen

Sie können den Prozess auch automatisieren. Dadurch können Sie mit einem Funktionsaufruf bei dem Sie die Matrix und den Lösungsvektor übergeben, die Gleichungen lösen lassen. Dies ist eventuell praktisch, wenn Sie eine Funktion in einem eigenem Paket bereitstellen wollen.

```
(%i1) M : matrix(
    [1, 2],
    [1, 2]
 )$
(%i2) v : matrix(
    [5],
    [5]
 )$
(%i3) matsolve (M, b) :=
(
  block ([n, a, c, gl, lsg],
    n : matrix_size(M)[1],
    lsglist : makelist( concat(x, i), i, 1, n),
    x : genmatrix( lambda( [i, j], concat(x, i) ), n, 1),
    /* x : columnvector [x1, x2, ..., xn] */
    c : M.x,
    gl : [],
    for i : 1 thru n do gl : append (gl, [ c[i, 1] = b[i, 1] ]),
    lsg : solve (gl, lsglist),
    return (lsg)
  )
)$
(%i4) matsolve (M, v);
0 errors, 0 warnings
solve: dependent equations eliminated: (2)
(%o4)                    [[x1 = 5 - 2 %r1, x2 = %r1]]
```

Abbildung 7.29: Der Code ist hier: 284

In den ersten beiden Zeilen werden M und v definiert. Dies wird hier ausnahmsweise und nicht empfehlenswerter Weise mit einem „$“ Zeichen am Ende gemacht. Dabei bekommen Sie keine Ausgabe und haben somit keine Kontrolle über Ihre Eingabe. Wir haben haben mehr Platz bei der Darstellung.

Dann erstellen wir die Funktion matsolve. Der werden M und b übergeben. M muss eine quadratische Matrix sein. Das wird nicht aus Übersichtsgründen hier jetzt nicht überprüft. Es gibt somit auch keine Fehlermeldung.

block $([lokaleVariablen], Anweisungen)$ Es wird ein Block erstellt. Zuerst wird angegeben, welche Variablen lokal genutzt werden.

n wird die Anzahl der Reihen der Matrix zugewiesen.

Dann wird die Liste lsglist erstellt. **makelist** erstellt aus Elementen eine Liste. Dazu nimmt die Variable i nacheinander die Werte 1 bis n an. Diese wird gebraucht, damit **solve** später weiss, wie die Variablen benannt sind. **concat** fügt zu jedem x den Wert der Variablen i zu, wobei i die Werte von 1 bis n annimt. Es wird folgende Liste erstellt: [x1, x2, ..., xn].

Die Matrix a ist dann ein Vektor, mit n Zeilen und 1 Spalte. Die Einträge werden durch die Funktion: **lambda ([i, j], concat (x, i))** berechnet. i nimmt nacheinander die Werte von 1 bis n an und j immer nur den Wert 1. **lambda** ist eine anonyme Funktion, mit den Variablen i und j. Bei der Berechnung des Rückgabewertes wird die Funktion **concat (x, i)** bemüht. Diese erzeugt wiederum ein x verbunden mit dem Wert von i. **genmatrix** erzeugt dann eine $(n \times 1)$-Matrix. Die Einträge sind dann x1, x2 usw. bis xn.

M.x erzeugt einen Vektor mit dem Gleichungsteil links vom Gleichheitszeichen. also hier:

$$\begin{pmatrix} x1 + 2x2 \\ x1 + 2x2 \end{pmatrix}$$

In der folgenden Schleife werden die Gleichungen werden jetzt Zeile um Zeile erzeugt. Die Gleichungen werden der Liste gl angehängt. Diese muss in Maxima einmal als Liste deklariert werden damit Maxima weiss, dass gl eine Liste ist. **append** fügt zwei Listen zusammen. gl ist eine Liste aus Gleichungen.

Im anschliessenden **solve** werden die Gleichungen in der Liste gl gelöst. Damit **solve** weiss, was die Variablen sind, sind die Variablennamen vorher in lsglist gespeichert.

Das Gaußsche Eleminationsverfahren mit echelon durchführen

echelon (*Matrix*) führt das Gaußsche Eleminationsverfahren durch, bis Sie eine obere Dreiecksmatrix haben.

```
(%i1) M : matrix( [1, 2], [1, 2] );
                                    [ 1  2 ]
(%o1)                               [      ]
                                    [ 1  2 ]
(%i2) v : matrix( [5], [5] );
                                      [ 5 ]
(%o2)                                 [   ]
                                      [ 5 ]
(%i3) C : addcol(M, v);
                                   [ 1  2  5 ]
(%o3)                              [         ]
                                   [ 1  2  5 ]
(%i4) echelon (C);
                                   [ 1  2  5 ]
(%o4)                              [         ]
                                   [ 0  0  0 ]
```

Abbildung 7.30: Der Code ist hier: 285

echelon (*M*) formt die Matrix M mit Hilfe des Gauß Algorithmus um, bis auf der Diagonalen Einsen (oder Nullen) stehen. **triangularize** (*M*) formt ebenfalls wie das Gaußsche Eleminationsverfahren um, aber auf der Hauptdiagonalen sind nicht notwendigerweise Einsen.

Hier muss man sich nun selbst um eine Lösung bemühen.

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \end{pmatrix} + r \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

Unangenehmer wird es, wenn es sich um eine (3×3)-Matrix handelt:

```
(%i1) M : matrix( [1, 2, 1], [1, 1, 2], [2, 3, 3] )$
(%i2) v : matrix( [8], [9], [17] )$
(%i3) C : addcol(M, v);
                                  [ 1  2  1  8  ]
                                  [             ]
(%o3)                             [ 1  1  2  9  ]
                                  [             ]
                                  [ 2  3  3  17 ]
(%i4) D : echelon (C);
                                [ 1  2   1    8  ]
                                [                 ]
(%o4)                           [ 0  1  - 1  - 1  ]
                                [                 ]
                                [ 0  0   0    0  ]
(%i5) T1 : matrix( [1, -2, 0], [0, 1, 0], [0, 0, 1] );
                                  [ 1  - 2  0 ]
                                  [           ]
(%o5)                             [ 0   1   0 ]
                                  [           ]
                                  [ 0   0   1 ]
(%i6) T1 . D;
                                [ 1  0   3   10  ]
                                [                ]
(%o6)                           [ 0  1  - 1  - 1 ]
                                [                ]
                                [ 0  0   0    0  ]
```

Abbildung 7.31: Der Code ist hier: 286

Nun müssen Sie noch selber von Hand weiter rechnen bis Sie oben links einen Block bestehend aus einer Einheitsmatrix erhalten:

Sie ziehen von der ersten Zeile das Doppelte der zweiten Zeile ab. Dies erhalten Sie, wenn Sie die Matrix T1 von links mit der Matrix D multiplizieren.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 10 \\ -1 \\ 0 \end{pmatrix} + r \begin{pmatrix} -3 \\ 1 \\ 1 \end{pmatrix}$$

Matrix-Verfahren mit Teilmatrizen

Für Liebhaber der Matrixschreibweise, bzw. wenn Sie eine Funktion bauen wollen, dann gibt es natürlich noch eine Möglichkeit nur mit Matrizen zu arbeiten.

Dabei werden wir mit Teilmatrizen arbeiten und damit spezielle Lösungen jeweils suchen.

Hier wird folgende Gleichung gelöst:

$$\begin{pmatrix} 1 & 2 & 1 \\ 1 & 1 & 2 \\ 2 & 3 & 3 \end{pmatrix} \vec{x} = \begin{pmatrix} 8 \\ 9 \\ 17 \end{pmatrix}$$

```
(%i1) M : matrix ( [1, 2, 1], [1, 1, 2], [2, 3, 3] );
                                  [ 1  2  1 ]
                                  [         ]
(%o1)                             [ 1  1  2 ]
                                  [         ]
                                  [ 2  3  3 ]
(%i2) v : matrix ( [8], [9], [17] );
                                    [ 8  ]
                                    [    ]
(%o2)                               [ 9  ]
                                    [    ]
                                    [ 17 ]
(%i3) /* Erstellen der erweiterten Matrix */
C : addcol(M, v);
                                [ 1  2  1  8  ]
                                [             ]
(%o3)                           [ 1  1  2  9  ]
                                [             ]
                                [ 2  3  3  17 ]
(%i4) /* Wenn det(M) null ist, gibt es mehrere Lsg. */
(%i4) determinant (M);
(%o4)                               0
```

Abbildung 7.32: Der Code ist hier: 287

Zuerst werden die Matrizen und die erweiterte Matrix definiert.

Im folgenden werden - damit die Übersicht erhalten bleibt - die vorherigen Teile jeweils mit **batchload** $(filename)$ geladen.

minor (*M, i, j*) gibt eine Teilmatrix von M zurück, bei der die i.-te Zeile und j.-te Reihe herausgenommen sind. M33 ist eine Teilmatrix von M und C33 ist eine Teilmatrix von C der erweiterten Matrix.

Die Determinante von M war null. D. h., dass es mehrere Lösungen der Gleichung: $M\vec{x} = \vec{v}$ gibt. Um nun eine spezielle Lösung zu finden, welche die Gleichung löst, suchen Sie eine Teilmatrix von M, deren Determinante ungleich null ist. Von der erweiterten Matrix suchen Sie dann eine Lösung der Gleichung: $C\vec{x} = 0$. Wenn in der letzten Zeile eine (-1) steht, dann haben Sie eventuell eine Lösung der ursprünglichen Gleichung $M\vec{x} = \vec{v}$. (Wenn dort keine (-1) steht, müssen Sie Werte des Vektor noch teilen.) Dies müssen Sie noch überprüfen. Es könnte ja auch sein, dass die Gleichung letztendlich gar keine Lösung hat.

```
(%i1) batchload
("MatrizenGleichungenMehrereLsgInGleichungLiebhaberTeil1.mac")$
(%i2) /* --    Zuerst wird eine spezielle Loesung gesucht -- */
/* Eine Teilmatrix mit einer von null verschiedenen Determinante */
M33 : minor (M, 3, 3);
                                    [ 1  2 ]
(%o2)                               [      ]
                                    [ 1  1 ]
(%i3) determinant (M33);
(%o3)                                 - 1
(%i4) /* Das Loeschen der 3. Reihe und der 3. Zeile ist in Ordnung */
C33 : minor (C, 3, 3);
                                  [ 1  2  8 ]
(%o4)                             [         ]
                                  [ 1  1  9 ]
(%i5) /* Die Spezielle Loesung wird in zwei Schritten erzeugt */
spezielle\ Loesung : args(nullspace (C33))[1];
0 errors, 0 warnings
                                    [ 10  ]
                                    [     ]
(%o5)                               [ - 1 ]
                                    [     ]
                                    [ - 1 ]
(%i6) spezielle\ Loesung [3][1] : 0 $
(%i7) spezielle\ Loesung;
                                    [ 10  ]
                                    [     ]
(%o7)                               [ - 1 ]
                                    [     ]
                                    [  0  ]
(%i8) M . spezielle\ Loesung;
                                    [ 8  ]
                                    [    ]
(%o8)                               [ 9  ]
                                    [    ]
                                    [ 17 ]
```

Abbildung 7.33: Der Code ist hier: 288

Nun werden die Vektoren gesucht, die die Gleichung lösen: $M\vec{x} = 0$.

```
(%i1) batchload
("MatrizenGleichungenMehrereLsgInGleichungLiebhaberTeil2.mac")$
0 errors, 0 warnings
(%i2) /* -- Dann werden die Loesungen der Gleichung Mx = 0 gesucht -- */
homLsg : args(nullspace(M))[1];
                                    [  3  ]
                                    [     ]
(%o2)                               [ - 1 ]
                                    [     ]
                                    [ - 1 ]
```

Abbildung 7.34: Der Code ist hier: 289

simp : false führt dazu, dass die Vektoren nicht zusammengerechnet werden. Wenn Sie weiterrechnen, sollten Sie **simp : true**, dass ist auch der voreingestellte Wert, angeben.

Sie können natürlich schon im Vorfeld wissen, wie viele Vektoren nullspace ergibt. Damit wissen Sie, wie viele Zeilen und Reihen Sie aus der Matrix M löschen müssen.

```
(%i1) M : matrix ([1, 1, 1], [0, 0, 0], [0, 0, 0]);
                                    [ 1  1  1 ]
                                    [         ]
(%o1)                               [ 0  0  0 ]
                                    [         ]
                                    [ 0  0  0 ]
(%i2) n : 3; /* Zeilen- und Spaltenanzahl */
(%o2)                                3
(%i3) dim1 : n - rank(M);
(%o3)                                2
(%i4) nullity (M);
0 errors, 0 warnings
(%o4)                                2
(%i5) echelon (M);
                                    [ 1  1  1 ]
                                    [         ]
(%o5)                               [ 0  0  0 ]
                                    [         ]
                                    [ 0  0  0 ]
```

Abbildung 7.35: Der Code ist hier: 290

Sie können entweder den Rang der Matrix mit **rank** (*M*) bestimmen, und diesen dann von der Anzahl der Reihen / Spalten abziehen, Sie benutzen die Funktion **nullity** (*M*) oder Sie zählen die Reihen, die nur aus Nullen bestehen.

7.4.5 Das Gaußsche Eleminationsverfahren

Mit Hilfe von Maxima können Sie auch von Hand das Gaußsche Eleminationsverfahren durchführen. Das ist zur Demonstration eventuell nützlich, besonders, um das Verfahren nach der Benutzung von **echelon** (*Matrix*) abzuschliessen.

Es gibt zwei Befehle:

- Sie können von einer Zeile (i) das Vielfache (theta) einer anderen Zeile (j) abziehen: Der Befehl dazu ist **rowop** (*M, i, j, theta*). In M wird die i.-te Zeile ersetzt durch die i.-te Zeile minus $theta$ mal der j.-ten Zeile:
$$i < -i - theta \cdot j$$
- Zeilen können Sie vertauschen durch **rowswap** (*M, i, j*)

Damit Sie das Lösungsverfahren wie gewohnt durchführen können, benötigen Sie (mindestens) zwei Funktionen: teileZeile und gauss.

1. teilezeile $(M, i, theta)$ Diese Funktion teilt die i.-te Zeile der Matrix M durch theta.
2. gauss $(M, ifaktor, i, jfaktor, j)$. Es wird eine Matrix zurückgegeben, bei der die i.-te Zeile die Differenz darstellt aus den Zeilen i und j, welche mit den Faktoren multipliziert wurden.
neue i.-te Zeile = ifaktor · i.te- Zeile - jfaktor · j.te-Zeile.

```
(%i1) load (eigen)$
(%i2) /* Ri * theta */
teileZeile (M, i, theta) :=
(
  block([B],
    B : copymatrix (M),
    for j : 1 thru matrix_size (M)[1] do
      B[i][j] : B[i][j] / theta,
    return (B)
  )
)$
(%i3) /* ifaktor * i - jfaktor * j */
gauss (M, ifaktor, i, jfaktor, j) :=
(
  block ([B, C],
    /* Die i.-te Zeile wird mit ifaktor multipliziert */
    B : rowop (M, i, i, 1 - ifaktor),
    C : rowop (B, i, j, jfaktor),
    return (C)
  )
)$
```

Abbildung 7.36: Der Code ist hier: 291

Im ersten Teil werden die Funktionen eingeführt. Mit **load** (*eigen*) wird die Funktion **columnvector** (*Liste*) zur Verfügung gestellt.

In der Funktion **teileZeile** wird zuerst ein **block** (. . .) erstellt. die Matrix B ist eine lokale Variable und ist zu Anfang eine direkt Kopie der Matrix M, welche der Funktion übergeben worden ist. Dies soll aber nicht verändert werden.

Jetzt wird eine Schleife erstellt, in der j nacheinander die Werte von 1 bis zur Matrixgröße annimmt. (M muss eine quadratische Matrix sein). Jetzt werden die einzelnen Elemente der Zeile durch theta geteilt und wieder an die Stelle geschrieben.

In der Funktion **gauss** werden zwei lokale Matrizen deklariert (B und C). In der 1. Zeile wird mit **rowop** die i.-te Zeile mit ifaktor multipliziert. Das ist etwas „trickiger“ als mit einer Schleife.

rowop $(M, i, i, 1 - ifaktor)$

$$\begin{aligned} \text{neu } R_i &= R_i - (1 - ifaktor) \cdot R_i \\ &= R_i - R_i + ifaktor \cdot R_i \\ &= ifaktor \cdot R_i \end{aligned}$$

C wird dann aus B berechnet, indem die Zeile R_i von C (CR_i) folgendermaßen aus den Zeilen der Matrix B gebildet wird:

$$CR_i = BR_i - jfaktor \cdot BR_j$$

```
(%i1) batchload ("MatrizenGaussTeil1.mac")$
(%i2) M : matrix ([1, 2], [2, 2]);
                                    [ 1  2 ]
(%o2)                               [      ]
                                    [ 2  2 ]
(%i3) /* lsg : 1, 3 */
v : columnvector ([7, 8]);
                                      [ 7 ]
(%o3)                                 [   ]
                                      [ 8 ]
(%i4) C : addcol (M, v);
                                   [ 1  2  7 ]
(%o4)                              [         ]
                                   [ 2  2  8 ]
(%i5) C1 : gauss (C, 1, 2, 2, 1);
0 errors, 0 warnings
                                [ 1   2    7  ]
(%o5)                           [             ]
                                [ 0  - 2  - 6 ]
(%i6) C2 : gauss (C1, 1, 1, -1, 2);
                                [ 1   0    1  ]
(%o6)                           [             ]
                                [ 0  - 2  - 6 ]
(%i7) C3 : teileZeile (C2, 2, -2);
                                 [ 1  0   1  ]
(%o7)                            [           ]
                                 [ 0  1  - 6 ]
```

Abbildung 7.37: Der Code ist hier: 292

7.5 Eigenwerte und Eigenvektoren

Sie haben eine Matrix M und möchten ihre Eigenwerte und Eigenvektoren bestimmen.

$$M = \begin{pmatrix} 27 & -4 \\ 6 & 13 \end{pmatrix}$$

```
(%i1) B : ident(2);
                                  [ 1  0 ]
(%o1)                             [      ]
                                  [ 0  1 ]
(%i2) eigenvalues (B);
(%o2)                            [[1], [2]]
(%i3) eigenvectors (B);
(%o3)                 [[[1], [2]], [[1, 0], [0, 1]]]
(%i4) M : matrix (
    [27, -4],
    [6, 13]
    )$
(%i5) eigenvalues (M);
(%o5)                          [[15, 25], [1, 1]]
(%i6) [werte, vek] : eigenvectors (M);
                                                        1
(%o6)            [[[15, 25], [1, 1]], [[1, 3]], [[1, -]]]
                                                        2
(%i7) werte;
(%o7)                          [[15, 25], [1, 1]]
(%i8) vek;
                                        1
(%o8)                         [[1, 3]], [[1, -]]]
                                        2
```

Abbildung 7.38: Der Code ist hier: 293

Das Paket „eigen“ wird automatisch bei der Benutzung von **eigenvalues** (*Matrix*) oder **eigenvectors** (*Matrix*) geladen.

Zum leichteren Verständnis der beiden Befehle **eigenvalues** (*Matrix*) und **eigenvectors** (*Matrix*) untersuchen wir zwei Matrizen: Eine (2×2)-Einheitsmatrix und M.

1. Die (2×2)-Einheitsmatrix:

   Um die Eigenwerte zu bestimmen müssen Sie die Nullstellen des entsprechenden characteristischen Polynoms bestimmen:

   $$(1 - \lambda)^2 = 0$$

   $\lambda = 1$ ist ein Eigenwert und zwar der einzige. Die Vielfachheit des Eigenwertes bestimmt sich durch den Exponenten der Gleichung und ist 2. Daher gibt der Befehl **eigenvalues**(B) zwei Listen: [1] und [2]. Die erste Liste besteht in diesem Fall aus einem Element und enthält den Eigenwert. Die zweite Liste gibt die Vielfachheit dieses Eigenwertes an.

   Die Eigenvektoren von B sind gerade $\binom{1}{0}$ und $\binom{0}{1}$.

2. M:

   eigenvectors (*Matrix*) gibt Ihnen zwei Listen. In der ersten Liste erhalten Sie die Ausgabe von **eigenvalues** (*Matrix*
   In der zweiten Liste erhalten Sie eine Liste der dazugehörigen Eigenvektoren.

eigenvalues (*Matrix*) gibt Ihnen die Eigenwerte (15 und 25) in einer ersten Liste und deren Vielfachheiten (jeweils 1) in einer zweiten Liste.

eigenvectors (*M*) gibt Ihnen die Eigenwerte 15 und 25, welche jeweils die Vielfachheit 1 haben. Die Eigenvektoren von M sind $\binom{1}{3}$ und $\binom{1}{0{,}5}$.

7.6 Bestimmen der stationären Lösung / des Fixpunktes

Die Gleichung:

$$M\vec{x} = \vec{x}$$

ergibt je nach Zusammenhang den Fixpunkt oder die stationäre Lösung. Nicht zu jeder Matrix lässt sich ein solcher Vektor $\vec{x}$ finden, der die Gleichung löst. Aber z. B. haben Stochastische Matrizen immer eine stationäre Lösung.

7.6.1 Lösen der Gleichung

```
(%i1) M : matrix (
    [1/5,  2/5, 1/2],
    [1/10, 1/5, 0  ],
    [7/10, 2/5, 1/2]
    )$
(%i2) M, numer : true;
                                [ 0.2  0.4  0.5 ]
                                [               ]
(%o2)                           [ 0.1  0.2   0  ]
                                [               ]
                                [ 0.7  0.4  0.5 ]
(%i3) M - identfor (M), numer : true;
0 errors, 0 warnings
                              [ - 0.8   0.4    0.5  ]
                              [                     ]
(%o3)                         [  0.1   - 0.8    0   ]
                              [                     ]
                              [  0.7    0.4   - 0.5 ]
(%i4) /* loest Gleichung (M - E) x = 0 */
l : nullspace (M - identfor(M));
                                     [ - 40 ]
                                     [      ]
(%o4)                          span([ - 5  ])
                                     [      ]
                                     [ - 60 ]
(%i5) first (l);
                                   [ - 40 ]
                                   [      ]
(%o5)                              [ - 5  ]
                                   [      ]
                                   [ - 60 ]
```

Abbildung 7.39: Der Code ist hier: 294

Die Matrix M wird hier in Brüchen angegeben, damit es nicht so viele Ausgaben bei der Umrechnung von Dezimalzahlen in Brüchen gibt. Sonst werden die Umrechnungen und ihre Fehler angezeigt durch:
rat: replaced -0.8 by -4/5 = -0.8
Sie erhalten aber die Ausgabe in Dezimalzahlen, wenn Sie hinter der Matrix für diese Zeile das Flag: „numer : true“ setzen.

Um die stationäre Lösung zu berechnen müssen Sie auf der Hauptdiagonalen bei den Elementen jeweils 1 abziehen. **identfor** (*M*) gibt Ihnen eine Einheitsmatrix, welche dieselben Reihen und Spalten hat wie M. Wenn Sie **identfor** benutzen, dann sollten Sie sicherstellen, dass M eine quadratische Matrix ist. Sonst gibt es unvorhergesehene Effekte, weil die Einsen dann unerwartet verteilt sind.

Durch **nullspace** (*M*) lösen Sie die Gleichung $M\vec{x} = 0$ der Matrix M, die Sie an **nullspace** (*Matrix*) übergeben.

Sie erhalten dann alle Vektoren, die diese Gleichung lösen mit der Angabe **span** (*Matrix*) versehen. **span** (*Matrix*) bedeutet dass alle Vielfache diese Gleichung lösen. Leider können Sie nicht direkt darauf zugreifen.

Auf die Vektoren können Sie mit first, second usw. zugreifen.

7.6.2 Lösen mit Hilfe von Eigenvektoren

Stochastische Matrizen haben eine stationäre Lösung, also ist genau ein Eigenwert immer eins. Allgemein bestimmen sich Eigenwerte und deren zugehörigen Eigenvektoren aus folgender Gleichung:

$$M\vec{x} = \lambda\,\vec{x}$$

Dabei ist λ ein Eigenwert und $\vec{x}$ der zugehörige Eigenwert. Im Falle eines stationären Zustandes ist $\lambda = 1$.

```
(%i1) load (eigen)$
(%i2) M : matrix (
    [1/5,  2/5, 1/2],
    [1/10, 1/5, 0  ],
    [7/10, 2/5, 1/2]
    )$
(%i3) M, numer : true;
                               [ 0.2  0.4  0.5 ]
                               [                ]
(%o3)                          [ 0.1  0.2   0  ]
                               [                ]
                               [ 0.7  0.4  0.5 ]
(%i4) [werte, vek] : eigenvectors (M);
           sqrt(21) + 1  sqrt(21) - 1
(%o4) [[[- ------------, ------------, 1], [1, 1, 1]],
                 20            20
      sqrt(21) - 5    sqrt(3) sqrt(7) - 3
[[[1, ------------, - -------------------]],
           2                   2
       sqrt(21) + 5  sqrt(3) sqrt(7) + 3         1  3
[[1, - ------------, -------------------]], [[1, -, -]]]
            2                   2                8  2
(%i5) werte[1][3]; /* Eigenwert der stat. Lsg */
(%o5)                                  1
(%i6) /* Eigenvektor der stat. Lsg */
(%i6) vek[3][1];
                                       1  3
(%o6)                             [1, -, -]
                                       8  2
```

Abbildung 7.40: Der Code ist hier: 295

Sie können das Paket **eigen** nachladen. Anschliessend weisen Sie die Eigenvektoren und Eigenwerte der Liste werte und vek zu. **eigenvectors** (*Matrix*) gibt Ihnen eine Liste bestehend aus Elementen zurück. Das erste Element besteht aus zwei Listen. Die erste Liste ist eine Liste der Eigenwerte von M. (hier: $-\frac{\sqrt{21}+1}{20}$, $\frac{\sqrt{21}-1}{20}$ und 1) Die zweite Liste gibt die Multiplizitäten / Vielfachheiten der Eigenwerte in der characteristischen Gleichung

an. [1] Da Sie drei verschiedene Eigenvektoren haben, ist deren Häufigkeit auch jeweils 1.
Die zweite Liste ist eine Liste der Eigenvektoren.

Sie nehmen sich den 3. Vektor mit dem Eigenwert 1 heraus. Sie könnten jetzt noch mit **covect** (*vek[3][1]*) einen Spaltenvektor erstellen.

[1] $$(x-3)(x-2)^3(x-5)^4 = 0$$

Die Vielfachheiten der Nullstellen sind gerade jeweils die Exponenten der in der folgenden Gleichung: $(x-3)(x-2)^3(x-5)^4 = 0$
Die Nullstelle 3 hat die Vielfachheit 1, die 2 die 3 und die 5 die 4.

Kapitel 8

Vektorrechnung

Vektoren werden in Maxima als 1-spaltige Matrizen aufgefasst. Das Paket vector_rebuild und das Paket eigen werden für verschiedene Funktionen benötigt. Das Paket vector_rebuild ist erzeugt unter anderem auch eine bessere Lesbarkeit bzw. Darstellung der Vektoren.

Viele Beispiele sind von Volker van Nek, den Autor des Paketes von vector_rebuild.

Die ausführliche Dokumentation dieses Paketes findet sich unter: share/vector/vector_rebuild.usg. Um den genauen Pfad auf Ihrem System zu finden, gehen Sie folgendermaßen vor:

1. Laden Sie das Paket mit
   load (vector_rebuild); Benutzen Sie ein Semikolon als Abschluss. Dann erhalten Sie den Pfad auf Ihrem System.

2. Verändern Sie dann bei dem ausgegebenen Pfad vector_rebuild.mac in vector_rebuild.usg und schauen sich diese Datei mit einem Programm Ihrer Wahl an.

vector_rebuild setzt verschiedene Optionen. Dadruch erfolgt z. B. nicht immer sofort eine Auswertung. Die Option vector_simp vereinfacht dann die Ausdrücke.

Aus Platzgründen wird die Zuweisung der Vektoren hier in der Regel mit einem $-Zeichen abgeschlossen.

8.1 Eingabe von Vektoren

Vektoren werden in Maxima als 1-spaltige Matrizen eingegeben. Der Vektor $\binom{1}{0}$:

```
(%i1) a : matrix ([1], [0]);
                                    [ 1 ]
(%o1)                               [   ]
                                    [ 0 ]
(%i2) load (vector_rebuild)$
(%i3) b : covect ([1, 0]);
                                    [ 1 ]
(%o3)                               [   ]
                                    [ 0 ]
```

Abbildung 8.1: Der Code ist hier: 296

Das Paket **load** (*vector_rebuild*) stellt den Befehl **covect** (*Liste*) zur Verfügung mit dem Sie einfach einen (1-spaltigen) Vektor eingeben können. **covect** (*Liste*) ist eine Kurzform für **columnvector** (*Liste*).

8.2 Rechnen mit Vektoren

Sie können mit Vektoren rechnen:

```
(%i1) load (vector_rebuild)$
(%i2) a : covect ([1,0])$
(%i3) b : covect ([2,1])$
(%i4) a + b;
                                  [ 2 ]   [ 1 ]
(%o4)                             [   ] + [   ]
                                  [ 1 ]   [ 0 ]
(%i5) a + b, vector_simp;
                                     [ 3 ]
(%o5)                                [   ]
                                     [ 1 ]
```

Abbildung 8.2: Der Code ist hier: 297

Laden Sie das Paket vector_rebuild.
Erst mit **vector_simp** wird dann der Vektor zum weiteren Gebrauch vereinfacht.

8.3 Columnvektoren umformen

8.3.1 Columnvektor zu einer Liste umformen

Sie haben einen Columnvektor und möchten gerne wieder eine Liste haben (z. B., um einen Punkt zeichnen zu lassen).

```
(%i1) load (vector_rebuild)$
(%i2) a : covect ([1, 2, 3])$
(%i3) b : covect ([2, 1, 2])$
(%i4) print ("a = ", a, " b = ", b)$
     [ 1 ]       [ 2 ]
     [   ]       [   ]
a =  [ 2 ]  b =  [ 1 ]
     [   ]       [   ]
     [ 3 ]       [ 2 ]
(%i5) dotproduct (a, b), vector_simp;
0 errors, 0 warnings
(%o5)                                10
(%i6) a . b, vector_simp;
(%o6)                                10
```

Abbildung 8.3: Der Code ist hier: 298

Mit **list_matrix_entries** (*e*)rhalten Sie wieder eine Liste. Sie können die Liste auch „manuell“ erstellen.

8.3.2 Columnvektor ganzzahlig machen

Sie können sowohl mit Listen als auch mit Columnvektoren arbeiten:

```
(%i1) load (vector_rebuild)$
(%i2) a : covect( [1/2, 1/3] )$
(%i3) a, vector_factor;
                                    1 [ 3 ]
(%o3)                               - [   ]
                                    6 [ 2 ]
(%i4) b : [1/2, 1/3];
                                     1  1
(%o4)                               [-, -]
                                     2  3
(%i5) b, vector_factor;
                                   1
(%o5)                              - [3, 2]
                                   6
```

Abbildung 8.4: Der Code ist hier: 299

8.4 Das Skalarprodukt

Das Skalarprodukt können Sie auf zwei Arten durchführen:

1. Sie bestimmen das Skalarprodukt mit **dotproduct** (*columnvector, columnvector*).
2. Sie bestimmen das Skalarprodukt mit dem Punktoperator und anschliessend mit Komma abgetrennt **vector_simp**

```
(%i1) load (vector_rebuild)$
(%i2) a : covect ([1, 2, 3])$
(%i3) b : covect ([2, 1, 2])$
(%i4) print ("a = ", a, " b = ", b)$
     [ 1 ]       [ 2 ]
     [   ]       [   ]
a =  [ 2 ]  b =  [ 1 ]
     [   ]       [   ]
     [ 3 ]       [ 2 ]
(%i5) dotproduct (a, b), vector_simp;
0 errors, 0 warnings
(%o5)                                10
(%i6) a . b, vector_simp;
(%o6)                                10
```

Abbildung 8.5: Der Code ist hier: 300

Sie laden das Paket „vector_rebuild“.
Dann bestimmen Sie mit **dotproduct** (*columnvector, columnvector*), vector_simp das Skalarprodukt:

$$1 \cdot 2 + 2 \cdot 1 + 3 \cdot 2 = 10$$

8.5 Bestimmen eines Normalenvektors

Sie haben zwei Vektoren $\vec{a}$ und $\vec{b}$ zu dem Sie einen Normalenvektor haben wollen. $\vec{a} = \left(\begin{smallmatrix}1\\2\\3\end{smallmatrix}\right)$ $\vec{b} = \left(\begin{smallmatrix}2\\1\\2\end{smallmatrix}\right)$
Hier werden drei Möglichkeiten vorgestellt einen Normalenvektor zu erhalten.

8.5.1 Das Kreuzprodukt

Sie möchten das Kreuzprodukt zwischen den beiden Vektoren $\vec{a}$ und $\vec{b}$ bilden:

$$\vec{n} = \vec{a} \times \vec{b} = \begin{pmatrix}1\\2\\3\end{pmatrix} \times \begin{pmatrix}2\\1\\2\end{pmatrix}$$

```
(%i1) load (vector_rebuild)$
(%i2) a : covect ([1, 2, 3])$
(%i3) b : covect ([2, 1, 2])$
(%i4) n : a~b;
                                    [  1  ]
                                    [     ]
(%o4)                               [  4  ]
                                    [     ]
                                    [ - 3 ]
```

Abbildung 8.6: Der Code ist hier: 301

Sie laden mit **load** (*vector_rebuild*) das Vektor-Paket, damit Sie den Tildenoperator zur Verfügung haben.

covect $\sim$ **covect** liefert das Kreuzprodukt. **covect** (*Liste*) ist ein Synonym für **columnvector** (*Liste*) und ergibt einen Vektor (1-spaltige Matrix).

8.5.2 Bestimmen des Normalenvektors mit einer Maxima Funktion

Maxima stellt Ihnen eine Funktion zur Verfügung. , so dass alle möglichen Normalenvektoren angegeben werden.

```
(%i1) load (eigen)$
(%i2) a : covect ([1, 2, 3])$
(%i3) b : covect ([2, 1, 2])$
(%i4) print ("a = ", a, " b = ", b)$
     [ 1 ]       [ 2 ]
     [   ]       [   ]
a =  [ 2 ]  b =  [ 1 ]
     [   ]       [   ]
     [ 3 ]       [ 2 ]
(%i5) basis : orthogonal_complement (a, b);
0 errors, 0 warnings
                                      [  1  ]
                                      [     ]
(%o5)                          span([  4  ])
                                      [     ]
                                      [ - 3 ]
(%i6) first (basis);
                                    [  1  ]
                                    [     ]
(%o6)                               [  4  ]
                                    [     ]
                                    [ - 3 ]
```

Abbildung 8.7: Der Code ist hier: 302

Mit **load** (*eigen*) laden Sie ein Paket, so dass Ihnen die Funktion columnvector zur Verfügung steht.

v_1, ..., v_n müssen entweder 1-spaltige Matrizen oder covect's bzw. columndvectoren sein. Dann gibt Ihnen **orthogonal_complement** (*v_1, ..., v_n*) Vektoren, deren Skalarprodukt zu den angegebenen null ist. Diese Vektoren ihrerseits bilden eine Basis mit der sich alle Vektoren bilden lassen, deren Skalarprodukt zu den gegebenen Vektoren v_1, ..., v_n null ist.

Da wir uns hier im dreidimensionalen Raum befinden und zwei Vektoren angeben wird nur ein Vektor zurückgegeben. Alle Vielfache dieses Vektors sind orthogonal zu den beiden gegebenen Vektoren. Dies wird durch **span** angezeigt.

8.5.3 Bestimmen des Normalenvektors durch Lösen von Gleichungen

Sie haben zwei Vektoren und suchen dazu einen Normalenvektor:

$$\vec{a} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} 2 \\ 1 \\ 2 \end{pmatrix}$$

Bei dieser Rechnung nutzen wir die Tatsache, dass das Skalarprodukt aus den gegebenen Vektoren und dem gesuchten Normalenvektor null ist:

$$n_1 + 2n_2 + 3n_3 = 0$$
$$2n_1 + n_2 + 2n_3 = 0$$

```
(%i1) load (eigen)$
(%i2) a : covect ([1, 2, 3])$
(%i3) b : covect ([2, 1, 2])$
(%i4) print ("a = ", a, " b = ", b)$
     [ 1 ]       [ 2 ]
     [   ]       [   ]
a =  [ 2 ]  b =  [ 1 ]
     [   ]       [   ]
     [ 3 ]       [ 2 ]
(%i5) M : addcol (a, b);
                                  [ 1  2 ]
                                  [      ]
(%o5)                             [ 2  1 ]
                                  [      ]
                                  [ 3  2 ]
(%i6) MT : transpose (M);
                                [ 1  2  3 ]
(%o6)                           [         ]
                                [ 2  1  2 ]
(%i7) basis : nullspace (MT);
0 errors, 0 warnings
                                    [  1  ]
                                    [     ]
(%o7)                          span([  4  ])
                                    [     ]
                                    [ - 3 ]
(%i8) n : first (basis);
                                  [  1  ]
                                  [     ]
(%o8)                             [  4  ]
                                  [     ]
                                  [ - 3 ]
```

Abbildung 8.8: Der Code ist hier: 303

Sie können zwei Gleichungen aufstellen und mit **solve** (*Gleichung*) lösen lassen oder mit Hilfe von Matrizen die Gleichungen lösen lassen:

$$\vec{a}^T \cdot \vec{n} = 0$$
$$\vec{b}^T \cdot \vec{n} = 0$$

Sie erstellen mit **addcol** (*M*, *v*) eine Matrix aus den Vektoren. Dann wird die transponierte der Matrix gebildet. Diese transponierte Matrix bildet gerade den linken Teil der obigen Gleichungen.

nullspace (*MT*) löst die Gleichung: $MT\,\vec{x} = 0$. Die Lösungsvektoren von nullspace geben bilden eine Basis mit Hilfe der alle Vektoren gebildet werden können, welche $MT\,\vec{x} = 0$ lösen. Da es sich hier um den dreidimensionalen Raum handelt und $\vec{a}$ und $\vec{b}$ linear unabhängig sind, so erhalten Sie nur einen Vektor.

span gibt an, dass alle Vielfache dieses Vektors die Gleichung: $MT\vec{n} = 0$ lösen.

Diese Lösung bildet gerade den Normalenvektor.

8.6 Länge eines Vektors bestimmen

Sie haben einen Vektor und wollen dessen Länge bestimmen.

Die Länge eines Vektors:

$$\vec{a} = \begin{pmatrix} 5 \\ 30 \\ 6 \end{pmatrix}$$

Die Länge $|\vec{a}|$ des Vektors $\vec{a}$ berechnen Sie dann wie folgt:

$$|\vec{a}| = \sqrt{\underbrace{5^2 + 30^2 + 6^2}_{\vec{a}\cdot\vec{a}}} = 31$$

```
(%i1) load (vector_rebuild)$
(%i2) a : covect ([5, 30, 6]);
                                    [ 5  ]
                                    [    ]
(%o2)                               [ 30 ]
                                    [    ]
                                    [ 6  ]
(%i3) |a|;
(%o3)                                31
```

Abbildung 8.9: Der Code ist hier: 304

Laden Sie das Paket vector_rebuild.

Dann können Sie den Betrag direkt berechnen lassen indem Sie den Vektor durch zwei senkrechte Striche einschliessen.

Beachten Sie, dass die Funktion **length** (*Liste*) die Anzahl der Elemente der Liste angibt. **length**(a) ergibt somit 3. Dies entspricht der Dimension und nicht der Länge des Vektors.

8.7 Geradengleichungen

In diesem Abschnitt wird das Aufstellen und Gleichsetzen von Geradengleichungen gezeigt.

```
(%i1) load (vector_rebuild)$
(%i2) g : [1,2] + t*[3,4];
(%o2)                          t [3, 4] + [1, 2]
(%i3) r : g, vector_simp;
(%o3)                         [3 t + 1, 4 t + 2]
(%i4) vector_rebuild (r, [t]);
(%o4)                          t [3, 4] + [1, 2]
```

Abbildung 8.10: Der Code ist hier: 305

Für weitere Rechnungen können Sie mit **vector_simp** (*d*)ie Gleichung zusammenfassen.

Mit **vector_rebuild** erstellen Sie wiederum die Gleichung.

```
(%i1) load (vector_rebuild)$
(%i2) g : [1,1] + t*[1,2];
(%o2)                          t [1, 2] + [1, 1]
(%i3) h : [0,5] + s*[2,1];
(%o3)                          s [2, 1] + [0, 5]
(%i4) g, t=4, vetor_simp;
(%o4)                          4 [1, 2] + [1, 1]
(%i5) gl : extract_equations ( g = h );
(%o5)                    [t + 1 = 2 s, 2 t + 1 = s + 5]
(%i6) algsys (gl, [t, s]);
(%o6)                          [[t = 3, s = 2]]
```

Abbildung 8.11: Der Code ist hier: 306

Einzelne Punkte können Sie ausgeben, in dem Sie mit einem Komma abgetrennt den Wert für den Parameter angeben.

Mit dem Befehl **extract_equations** erzeugen Sie Gleichungen, die Sie mit **solve** (*Gleichung, Var*) oder mit **algsys** (*[Gleichung], [Variablen]*) lösen lassen können.

8.8 Ebenengleichungen

8.8.1 Einsetzen in die Parameterform

Sie haben die Parameterform und möchten einzelne Punkte der Ebene ausrechnen.

$$E : \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + r \begin{pmatrix} 2 \\ 1 \\ 5 \end{pmatrix} + s \begin{pmatrix} 3 \\ -2 \\ 1 \end{pmatrix}$$

```
(%i1) load (vector_rebuild)$
(%i2) v ([args]) := covect (args)$
(%i3) powerdisp : true$
(%i4) x : v(1, 2, 3) + r * v(2, 5, 1) + s * v(3, -2, 1);
                          [ 1 ]     [ 2 ]     [  3  ]
                          [   ]     [   ]     [     ]
(%o4)                     [ 2 ] + r [ 5 ] + s [ - 2 ]
                          [   ]     [   ]     [     ]
                          [ 3 ]     [ 1 ]     [  1  ]
(%i5) x, r = 0, s = 0, vector_simp;
                                    [ 1 ]
                                    [   ]
(%o5)                               [ 2 ]
                                    [   ]
                                    [ 3 ]
(%i6) x, r = 0, s = 2, vector_simp;
                                   [  7  ]
                                   [     ]
(%o6)                              [ - 2 ]
                                   [     ]
                                   [  5  ]
```

Abbildung 8.12: Der Code ist hier: 307

Laden Sie das Paket vector_rebuild.

Definieren Sie sich eine Funktion namens v, der Sie die Werte für den Vektor übergeben können. Diese Funktion erstellt jeweils einen covect mit den übergebenen Werten.

Die Option powerdisp : true sorgt für eine aufsteigende Anordnung der Summe.

Sie setzen die Werte also ein, indem Sie den Vektor $\vec{x}$ mit r und s definieren. Dann beim Aufrufen von x mit Komma abgetrennt angeben, wie groß r und s jeweils sein sollen.

8.8.2 Von der Parameterform in die HNF

Gegeben ist eine Ebene und Sie möchten die HNF erstellen.

$$E : \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + r \begin{pmatrix} -25 \\ 2 \\ 0 \end{pmatrix} + s \begin{pmatrix} 0 \\ 2 \\ -5 \end{pmatrix}$$

```
(%i1) load (vector_rebuild)$
(%i2) v([args]) := covect(args)$
(%i3) b : v(-25, 2, 0)$
(%i4) c : v(0, 2, -5)$
(%i5) n : b~c;
                                    [ - 10  ]
                                    [       ]
(%o5)                               [ - 125 ]
                                    [       ]
                                    [ - 50  ]
(%i6) n : vector_factor(n), vector_factor_minus : true;
                                       [ 2  ]
                                       [    ]
(%o6)                               - 5 [ 25 ]
                                       [    ]
                                       [ 10 ]
(%i7) en : 1/|n| * n;
                                         [ 2  ]
                                    1    [    ]
(%o7)                             (- --) [ 25 ]
                                    27   [    ]
                                         [ 10 ]
(%i8) |en|;
(%o8)                                   1
```

Abbildung 8.13: Der Code ist hier: 308

Laden Sie das Paket vector_rebuild

Definieren Sie sich eine Funktion namens v, der Sie die Werte für den Vektor übergeben können.

Bilden Sie den Normalenvektor zu den Richtungsvektoren mit dem Tildenoperator. Mit der Option vector_factor wird ein gemeinsamer Teiler als Faktor vor dem Vektor geschrieben. Die Option vector_factor_minus minimiert die Minuszeichen.

en ist dann ein Normalenvektor mit der Länge 1.

Die HNF ergibt sich jetzt leicht:

$$E: \left[\vec{x} - \begin{pmatrix}1\\2\\3\end{pmatrix}\right] \cdot \frac{1}{135}\begin{pmatrix}10\\125\\50\end{pmatrix} = 0$$

8.8.3 Von der Normalenform zur Parameterform

Sie haben eine Normalenform und möchten aber eine Parameterform haben.

$$E: \left[\vec{x} - \begin{pmatrix}1\\2\\3\end{pmatrix}\right] \cdot \begin{pmatrix}2\\25\\10\end{pmatrix} = 0$$

```
(%i1) load (vector_rebuild)$
(%i2) powerdisp : true$
(%i3) v([args]) := covect(args)$
(%i4) a : v(1,2,3)$
(%i5) n : v(2, 25, 10)$
(%i6) oc : orthogonal_complement(n)$
0 errors, 0 warnings
(%i7) [b, c] : [first(oc), second(oc)]$
(%i8) x : a + r * b + s * c;
                        [ 1 ]     [ - 25 ]     [  0   ]
                        [   ]     [      ]     [      ]
(%o8)                   [ 2 ] + r [  2   ] + s [  10  ]
                        [   ]     [      ]     [      ]
                        [ 3 ]     [  0   ]     [ - 25 ]
```

Abbildung 8.14: Der Code ist hier: 309

Laden Sie das Paket vector_rebuild.

Durch *powerdisp : true* erreichen Sie eine Umstellung der Summe in aufsteigender Richtung.

orthogonal_complement (*covect*) gibt Ihnen eine Basis des Raumes, deren Vektoren alle senkrecht auf n stehen. Senkrecht zum Normalenvektor sind alle Richtungsvektoren der Ebene.

Im nächsten Schritt erstellen Sie aus den beiden Richtungsvektoren eine Liste (durch Umklammerung mit den rechteckigen Klammern: []) und weisen Sie b und c zu.

Damit ergibt sich die Parameterform:

$$E : \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} + r \begin{pmatrix} -25 \\ 2 \\ 0 \end{pmatrix} + s \begin{pmatrix} 0 \\ 10 \\ -25 \end{pmatrix}$$

8.8.4 Von der Normalenform zur Koordinatenform

Sie haben einen Normalenvektor und einen Punkt und möchten gerne eine Koordinatenform haben:

$$\vec{n} = \begin{pmatrix} 2 \\ 25 \\ 10 \end{pmatrix} \quad \vec{a} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

```
(%i1) load (vector_rebuild)$
(%i2) v([args]) := covect(args)$
(%i3) a : v(1, 2, 3)$
(%i4) n : v(2, 25, 10)$
(%i5) nf : x.n = a.n;
                                [ 2  ]   [ 1 ]   [ 2  ]
                                [    ]   [   ]   [    ]
(%o5)                       x . [ 25 ] = [ 2 ] . [ 25 ]
                                [    ]   [   ]   [    ]
                                [ 10 ]   [ 3 ]   [ 10 ]
(%i6) E : nf, x = v(x1, x2, x3), vector_simp;
(%o6)                        10 x3 + 25 x2 + 2 x1 = 82
(%i7) E2 : E / |n|;
                             10 x3 + 25 x2 + 2 x1   82
(%o7)                        -------------------- = --
                                       27           27
```

Abbildung 8.15: Der Code ist hier: 310

Laden Sie das Paket vector_rebuild. Zur deutlicheren Eingabe schreiben Sie eine Funktion namens v, der Sie die Werte für die Vektoren übergeben.

Mit *nf : x.n = a.n;* schreiben Sie zuerst die Formel auf, welche dann im nächsten Schritt mit x = (x1, x2, x3) gefüllt wird und durch vector_simp dann auch zusammengefasst.

Wenn Sie eine Koordinatenform mit einem auf der Länge 1 normierten Normalenvektor haben wollen, teilen Sie die vorherige Koordinatenform durch die Länge des Normalenvektors.

8.9 Skalarprodukt / Koordinatenform – Anordnung

Sie haben eine Koordinatenform und möchten diese aufsteigend geordnet in Maxima ausgegeben bekommen.

```
(%i1) vx : [x1, x2, x3];
(%o1)                            [x1, x2, x3]
(%i2) n : [1, 2, 3];
(%o2)                              [1, 2, 3]
(%i3) vx . n;
(%o3)                          3 x3 + 2 x2 + x1
(%i4) powerdisp : true;
(%o4)                                true
(%i5) vx . n;
(%o5)                          x1 + 2 x2 + 3 x3
(%i6) E : vx . n = 5;
(%o6)                        x1 + 2 x2 + 3 x3 = 5
```

Abbildung 8.16: Der Code ist hier: 311

powerdisp verändert die Anordung der Summe. Normalerweise wird eine Summe in absteigender Ordnung angezeigt. Standardmäßig ist **powerdisp** : false.

Wenn **powerdisp : true** gilt, dann wird die Summe in aufsteigender Ordnung angezeigt.

8.10 Ortskurve des Höhenschnittpunktes

Sie wollen die Ortskurve des Höhenschnittpunktes der Dreiecke bestimmen, bei denen die Spitze C auf einer Parallelen verschoben wird.

Vergleichen Sie auch Problem: 13.13 S. 161.

Die Höhe hc ergibt sich dann wie folgt:

$$hc = C + r \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Die Normale zu b (AC) ist nb. Berechnen Sie C-A und vertauschen Sie die beiden Elemente und multiplizieren Sie einen mit (-1):

$$nb = \begin{pmatrix} C_y - A_y \\ -1(C_x - A_x) \end{pmatrix}$$

Die Höhe hb steht senkrecht auf AC und geht durch den Punkt B:

$$hb = B + s \cdot nb$$

Der Höhenschnittpunkt ist gerade der Schnittpunkt der beiden Geraden.

```
(%i1) load (draw)$
(%i2) A : [0, 0]$
(%i3) B : [1, 0]$
(%i4)
dreieck(x) :=
  block ([],
    C : [x, 1],
    hc : [ C[1], C[2] + r ],
    nb : [(C - A )[2], (-C + A)[1]],
    hb : B + s * nb,
    lsg : solve ([hb[1] = hc[1], hb[2] = hc[2]], [r, s]),
    H : ev (hc, lsg),
    l : [points ([A, B]), points ([B, C]), points ([A, C])],
    l : append (l, [points ([A, H]), points ([B, H]), points ([C, H])]),
    return (l)
  )$
(%i5)
dreieck (x);
(%o5) [points([[0, 0], [1, 0]]), points([[1, 0], [x, 1]]),
                                                      2
points([[0, 0], [x, 1]]), points([[0, 0], [x, x - x ]]),
                           2                                2
points([[1, 0], [x, x - x ]]), points([[x, 1], [x, x - x ]])]
(%i6) H;
                                          2
(%o6)                              [x, x - x ]
(%i7) f : H[2];
                                        2
(%o7)                                 x - x
(%i8)
draw2d (
    file_name = "hoehenschnittpunktsOrtskurve",
    dimensions = [500, 500],
    terminal = jpg,
    xrange = [-1, 2], yrange = [-1, 1.5],
    points_joined = true,
    dreieck (0), dreieck (0.5), dreieck (1),
    color = red,
    explicit (f, x, -1, 2)
    )$
rat: replaced 0.5 by 1/2 = 0.5
rat: replaced -0.5 by -1/2 = -0.5
```

Abbildung 8.17: Der Code ist hier: 312

lappend (*Liste1*, *Liste2*) gibt eine zusammengefügte Liste der beiden Listen zurück. Die beiden Listen bleiben unverändert.

Rufen Sie die Funktion dreieck ohne einen speziellen Wert sondern mit x auf. Maxima rechnet auch dann den Höhenschnittpunkt aus und speichert ihn in H. H[1] ist der x-Wert, H[2] ist der y-Wert des Höhenschnittpunktes.

Dann werden drei Dreiecke und die Ortskurve gezeichnet.

Kapitel 9

Statistik

9.1 Daten einlesen

Sie wollen Daten aus einer mit Kommata unterteilten Tabelle auslesen und die 2. Spalte als Liste zur weiteren Verarbeitung ausgeben:

```
(%i1) b : read_nested_list (file_search ("statistik1.data"), comma);
(%o1)          [[Schmidt, 1500], [Meier, 2000], [Mueller, 2500]]
(%i2) b;
(%o2)          [[Schmidt, 1500], [Meier, 2000], [Mueller, 2500]]
(%i3) map (lambda ([x], x[2]), b);
(%o3)                          [1500, 2000, 2500]
```

Abbildung 9.1: Der Code ist hier: 313

Aus der Datei „statistikdurchschnitt2.data“ lesen Sie die mit einem Komma unterteilten Daten mit **read_nested_list** (*Quelle, Trennzeichen*).

b; gibt Ihnen die eingelesenen Daten aus.

Mit **map (f, Ausdruck 1, . . . , Ausdruck n** wird eine Liste mit n Elementen erstellt, wobei die Funktion f auf die einzelnen Ausdrücke ausgeführt wird. Jede Anwendung auf den Ausdruck erzeugt einen Listeneintrag.

In diesem Fall wird die Funktion f nur auf b angewendet.

lambda (*[x], ldots*) ist eine anonyme Funktion, mit dem übergebenem Wert x. Hier wird jetzt angenommen, dass x eine Liste ist und der 2. Wert der Liste wird zurückgegeben. Das ist dann aber gerade die 2. Spalte.

Sollten die einzelnen Werte in der Datei durch ein spezielles Zeichen getrennt sein, so können Sie das Trennzeichen angeben. Trennzeichen können sein: „comma“, „semicolon“, „space“ oder „pipe“ für |.

Wenn die Werte getrennt sind durch Leerzeichen, können Sie das Trennzeichen auch weg lassen: **read_nested_list** (*Quelle*).

Quelle kann ein Dateiname oder ein Eingabefluss sein.

9.2 Berechnen des Durchschnitts

Laden Sie mit **load** (*descriptive*) das entsprechende Paket. Dann steht Ihnen der Befehl **mean** () zur Verfügung.

9.2.1 Berechnen des Durchschnitts von Listenelementen

```
(%i1) load (descriptive)$
(%i2) mean ([1, 2]), numer; /* Berechnet den Durchschnitt der Listenelemente
*/
(%o2)                                  1.5
(%i3) map (mean, [[1, 2, 3], [4, 5]]);
                                        9
(%o3)                                [2, -]
                                        2
(%i4) a : [1, 2, 3];
(%o4)                               [1, 2, 3]
(%i5) b : [4, 5];
(%o5)                                [4, 5]
(%i6) map (mean, [a, b]);
                                        9
(%o6)                                [2, -]
                                        2
```

Abbildung 9.2: Der Code ist hier: 314

Ein mit Komma abgetrenntes **numer** garantiert, dass Sie eine Dezimalzahl erhalten. Sonst erhalten Sie Brüche.

map (*f, Ausdruck, . . . , Ausdruck*) wendet die Funktion *f* hier also **mean** () auf die Ausdrücke an. Hier sind die Ausdrücke Listen, deren jeweiliger Durchschnitt berechnet wird.

9.2.2 Berechnen des Durchschnitts bei Tabellen

Sie haben in einer Datei eine Tabelle in der in der 1. Zeile Namen stehen und davon abgetrennt in der 2. Spalte Zahlen.

Schmidt, 1500
Meier, 2000
Mueller, 2500

Sie wollen nun den Durchschnitt der 2. Spalte errechnen lassen:

```
(%i1) load (descriptive);
(%o1) /usr/share/maxima/5.22.1/share/contrib/descriptive/descriptive.mac
(%i2) b : read_nested_list (file_search ("statistikdurchschnitt2.data"),
comma);
(%o2)          [[Schmidt, 1500], [Meier, 2000], [Mueller, 2500]]
(%i3) b;
(%o3)          [[Schmidt, 1500], [Meier, 2000], [Mueller, 2500]]
(%i4) map (lambda ([x], x[2]), b);
(%o4)                          [1500, 2000, 2500]
(%i5) mean (map (lambda ([x], x[2]), b));
(%o5)                                  2000
```

Abbildung 9.3: Der Code ist hier: 315

Sie laden mit **load** (*descriptive*) das Paket, damit Sie **mean** () benutzen können.

Aus der Datei „statistikdurchschnitt2.data“ lesen Sie die mit einem Komma unterteilten Daten.

b; gibt Ihnen die eingelesenen Daten aus.

Zur Verdeutlichung ist **map** (*lambda ([x], x[2]), b*); ausgegeben. Sie wollen die 2. Spalte als Liste bekommen. Dazu wird eine Liste mit **map** () erstellt. Das 1. Argument von **map** (*f, Ausdruck . . . Ausdruck*) ist eine Funktion, welche auf alle rechts davon stehenden Ausdrücke ausgeführt wird. In diesem Fall nur auf b. Von jeder Liste in b wird nun das 2. Element ausgegeben.

Wenn Sie den Durchschnitt der 2. Spalte berechnen wollen, dann übergeben Sie die 2. Spalte als Liste, wie oben beschrieben, der Funktion **mean** ().

9.3 Varianz oder Streuung bestimmen

Sie wollen die Varianz bestimmen.

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Diese Varianz berechnen Sie mit **var** (*Liste*). Die dazugehörige Standardabweichung berechnen Sie mit **std** (*Liste*).

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

Diese Varianz berechnen Sie mit **var1** (*Liste*). Die dazugehörige Standardabweichung berechnen Sie mit **std1** (*Liste*).

Die Datei „statistikvarianz.data“:

1 2 3 4 5

```
(%i1) load (descriptive)$
(%i2) data : read_list (file_search ("statistikvarianz.data"))$
(%i3) var (data), numer;
(%o3)                                2
(%i4) std (data), numer;
(%o4)                        1.414213562373095
```

Abbildung 9.4: Der Code ist hier: 316

Sie lesen eine Liste ein mit **read_list** (*Quelle*). Anschliessend berechnet **var** (*Liste*) dann die Varianz. **std** (*Liste*) berechnet die Standardabweichung.

9.4 Spannweite bzw. kleinster und größter Wert

Sie haben einen Datensatz in einer Datei und wollen darin den größten und kleinsten Wert, bzw. die Spannweite bestimmen.

Die Datei „statistikspannweite.data“:

1 2 3 4 5

```
(%i1) load (descriptive)$
(%i2) data : read_list (file_search ("statistikspannweite.data"))$
(%i3) data;
(%o3)                          [1, 2, 3, 4, 5]
(%i4) minimum : smin (data);
(%o4)                                  1
(%i5) maximum : smax (data);
(%o5)                                  5
(%i6) spannweite : maximum - minimum;
(%o6)                                  4
(%i7) spannweite : range (data);
(%o7)                                  4
```

Abbildung 9.5: Der Code ist hier: 317

smin (*Liste*) und **smax** (*Liste*) liefern den kleinsten Wert und den größten Wert der Liste.
Die Spannweite erhalten Sie durch einfach Subtraktion oder durch **range** (*Liste*).

9.5 Quantile und Median

Sie wollen die Quantile eines Datensatzes und den Median für einen Boxplot berechnen lassen.

```
(%i1) load (descriptive)$
(%i2) data : read_list (file_search ("statistikquantil.data"))$
(%i3) data;
(%o3)                  [1, 2, 3, 4, 5, 2, 3, 1]
(%i4) sort (data);
(%o4)                  [1, 1, 2, 2, 3, 3, 4, 5]
(%i5) quantile (data, 1/2), float;
(%o5)                                 2.5
(%i6) quantile (data, 1/4), float;
(%o6)                                1.75
(%i7) quantile (data, 3/4), float;
(%o7)                                3.25
(%i8) median (data), float;
(%o8)                                 2.5
(%i9) qrange (data), float;
(%o9)                                 1.5
```

Abbildung 9.6: Der Code ist hier: 318

Die Liste muss nicht sortiert sein. Zur besseren Übersicht sind mit **sort** (*Liste*) die Werte sortiert.

quantile (*Liste*, *Wert*) gibt Ihnen das Quantil der Liste zurück. Mit Wert können Sie eintragen, welches Quantil gesucht ist. Gängige Quantile sind 0.25 bzw. 0.75. Beachten Sie den Punkt statt des Kommas bei der Eingabe.

quantile (*Liste, 0.5*) ist mit dem **median** (*Liste*) identisch.

Den Insterquartilsabstand können Sie angeben lassen durch **qrange** (*Liste*). Dieser Abstand ist identisch zu: **quantile** (*Liste, 0.75*) - **quantile** (*Liste, 0.25*).

9.6 Darstellung eines Datensatzes

In diesem Abschnitt werden Darstellungsmöglichkeiten von Datensätzen vorgestellt:

1. Histogramm
2. Boxplot
3. Streudiagramm

9.6.1 Ein Histogramm erstellen

Die Funktion **histogramm ()** erzeugt ein Histogramm. Sie können entweder eine Liste oder eine 1-Spaltige (oder auch 1-Zeilige) Matrix übergeben. Wahlweise mit Optionen.

- histogram (liste);
- histogram (liste, optionen);
- histogram (matrix);
- histogram (matrix, optionen);

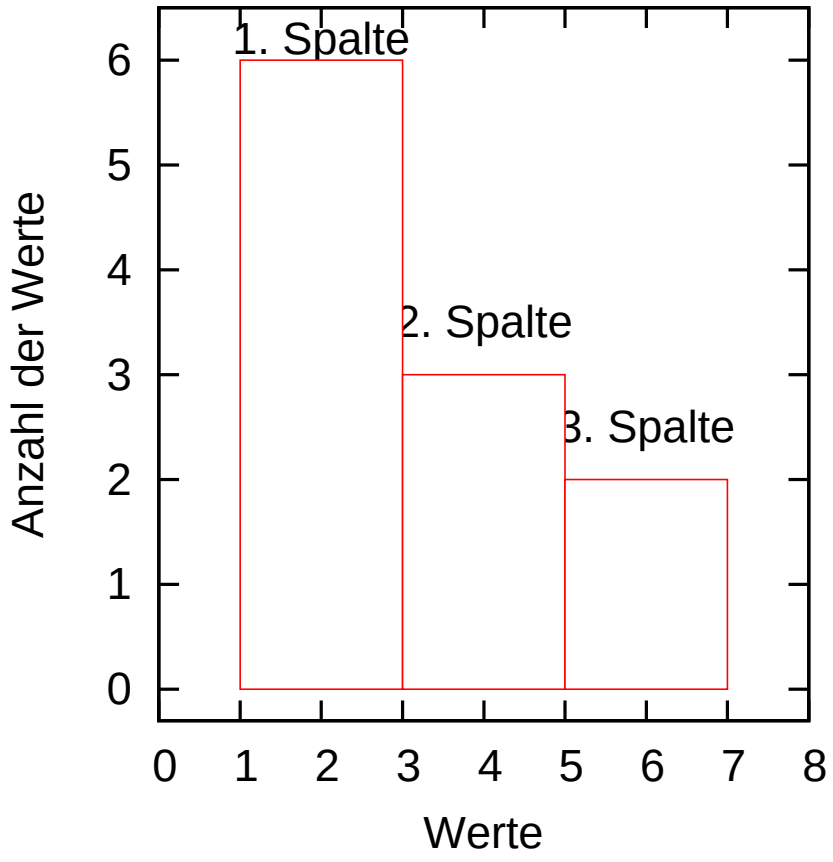
Optionen werden mit Kommata abgetrennt. Sie können die Optionen von **draw** verwenden. Beispiele für Optionen:

- nclasses = 10 (Standardwert)
  Anzahl der Klassen im Histogramm. nclasses muss größer als 2 sein.
- title = “Titel“,
- xlabel = “Einheit der x-Achse“,
- ylabel = “Einheit der y-Achse“,
- fill_color = grey,
- fill_density = 0.6,
- user_preamble.
  Mit dieser Option können Sie direkt Gnuplot-Befehle schreiben.

  Definieren Sie dazu erst eine Liste mit den Gnuplot-Befehlen und übergeben Sie diese dann mit **user_preamble =**

```
(%i1) load (descriptive)$
(%i2) data : read_list (file_search ("darstellung.data"))$
(%i3) sort (data);
(%o3)                  [1, 1, 2, 2, 3, 3, 4, 5, 5, 6, 7]
(%i4) my_gnuplot_preamble : [
(%i4)     "set label '1. Spalte' at 2, 6.2 center",
(%i4)     "set label '2. Spalte' at 4, 3.5 center",
(%i4)     "set label '3. Spalte' at 6, 2.5 center",
(%i4)     "set yrange [:6.5]"
(%i4)      ]$
(%i5) histogram (
    data,
    nclasses = 3,
    xlabel = "Werte",
    ylabel = "Anzahl der Werte",
    terminal = jpg,
    file_name =  "histogramm",
    user_preamble = my_gnuplot_preamble,
    dimensions = [500, 500]
    )$
```

Abbildung 9.7: Der Code ist hier: 319



load (*descriptive*) lädt das Paket descriptive für den Befehl **histogram** ().

In data werden die Daten aus der Datei „darstellung.data“ eingelesen. Zur besseren Übersicht werden die Daten sortiert ausgegeben.

Nun wird eine Liste namens my_gnuplot_preamble definiert, welche direkte Gnnuplot Befehle enthält.

set label ’1. Spalte’ at 2, 6.2 center

Es wird ein Label in die Zeichnung eingefügt bei den Koordinaten 2, 6.2. Durch den Zusatz center wird das Label um diese Koordinaten zentriert.

Sie übergeben dem Befehl **histogram** () zuerst die Daten und dann die Optionen:

- nclasses = 3
  Es werden drei Säulen gebildet.
- xlabel gibt die Beschriftung unterhalb der x-Achse an.
- ylabel gibt die Beschriftung an der y-Achse an.
- yrange gibt die Spannweite des Bildes in y-Richtung an. Der erste Wert bleibt unverändert, so wie die automatische Steuerung von Gnuplot das vorsieht. Der obere Wert wird hier dann neu gesetzt und vergrößert, so dass oben mehr Platz für die Schrift vorhanden ist.
- terminal gibt das Ausgabeformat an.
  Andere Ausgabeformate sind der Bildschirm (Standard), png, eps, pdf, wxt (wxwidget Fenster).

  Andere Ausgabeformate können mit Gnuplot angegeben werden.
- file_name gibt den Dateinamen für die Ausgabe an. Geben Sie den Dateinamen ohne die Endung an, weil sonst ein zusätzliches „jpg2„ an dem Dateinamen angehängt wird.
- user_preamble übergibt die Gnuplot Befehle direkt.

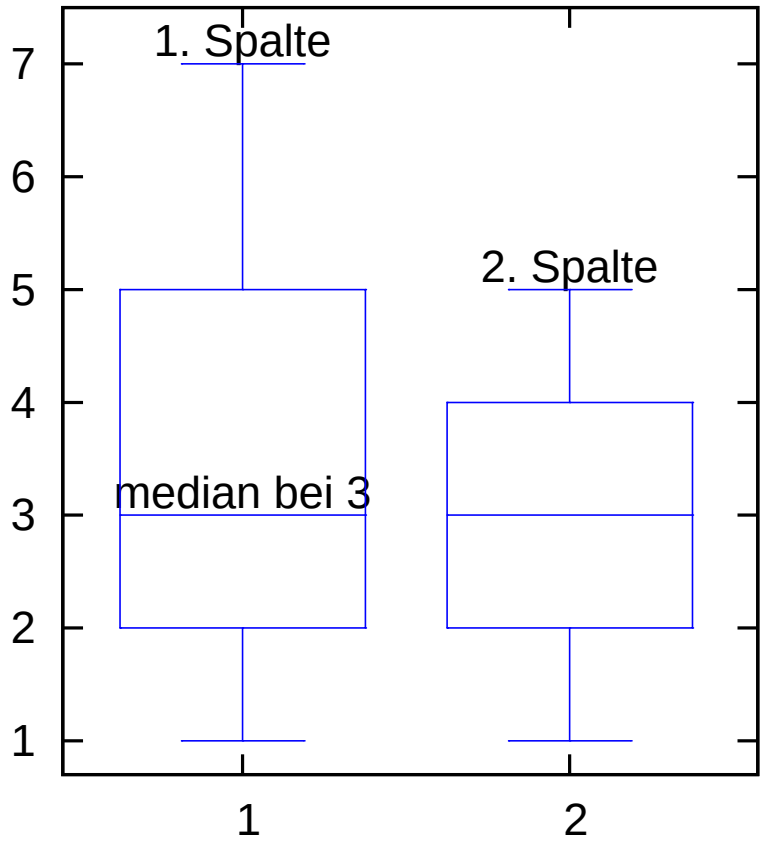
9.6.2 Einen Boxplot erstellen

Die Funktion **boxplot ()** erzeugt einen Boxplot. Sie können entweder eine Liste, eine Matrix oder eine Liste von Liste übergeben.

Dadurch sind Sie in der Lage verschiedene Verteilungen mit Hilfe eines Boxplots zu vergleichen.

```
(%i1) load (descriptive)$
(%i2) data : read_list (file_search ("darstellung.data"))$
(%i3) data2 : read_list (file_search ("darstellung2.data"))$
(%i4) sort (data);
(%o4)                  [1, 1, 2, 2, 3, 3, 4, 5, 5, 6, 7]
(%i5) sort (data2);
(%o5)                        [1, 2, 2, 3, 4, 4, 5]
(%i6)
load (distrib)$
(%i7) med1 : median (data);
(%o7)                                  3
(%i8) str1 : concat ("set label 'median bei ", string (med1));
(%o8)                        set label 'median bei 3
(%i9) str1 : concat (str1, "' at 1, 3.2 center");
(%o9)              set label 'median bei 3' at 1, 3.2 center
(%i10)
my_gnuplot_preamble : [
    str1,
    "set label '1. Spalte' at 1, 7.2 center",
    "set label '2. Spalte' at 2, 5.2 center",
    "set yrange [:7.5]"
     ]$
(%i11)
boxplot ([data, data2],
    user_preamble = my_gnuplot_preamble,
    terminal = jpg,
    file_name =  "boxplot",
    dimensions = [500, 500]
    )$
```

Abbildung 9.8: Der Code ist hier: 320



load (*descriptive*) stellt Ihnen die Funktion **boxplot** () zur Verfügung. **load** (*distrib*) stellt Ihnen die Funktion **median** () zur Verfügung.

Die Daten werden eingelesen und zur besseren Ansicht sortiert.

Um einen berechneten Wert, nämlich den Median, in einem Label zu schreiben, ist es notwendig erst eine Zeichenkette (String) zu erzeugen. Der Befehl **string** (*Ausdruck*) wertet den Ausdruck aus und erzeugt eine Zeichenkette. **concat** (*Zeichenkette, Zeichenkette*) hängt die beiden Zeichenketten aneinander. An gnuplot wird also folgende Zeichenkette übergeben:
set label 'median bei 3' at 1, 3.2

Dann erfolgt der Aufruf von **boxplot** (*[data, data2], Optionen*). Von beiden Verteilungen werden Boxplots erstellt. Weitere Optionen hier sind, dass die Ausgabe im jpg-Format erfolgen soll und in die Datei „boxplot.jpg“. Die Dateiendung wird automatisch angehängt.

9.7 Den Binomialkoeffizient bestimmen

Sie wollen den Binomialkoeffizienten zu einem gegebenen n und k bestimmen.

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$

Benutzen Sie den Befehl **binomial** (*n, k*).

```
(%i1) binomial (3, 2);
(%o1)                                3
```

Abbildung 9.9: Der Code ist hier: 321

9.8 Binomialverteilung: genau k mal

Sie haben eine binomialverteilte Größe (n, p). Sie wollen die Wahrscheinlichkeit wissen, dass das Zufallselement genau gleich k mal vorkommt.

$$B(n,p)(X=k) = \binom{n}{k}\, p^k\, q^{n-k}$$

```
(%i1) load (distrib)$
(%i2) pdf_binomial (3, 5, 0.2);
(%o2)                        .05120000000000002
```

Abbildung 9.10: Der Code ist hier: 322

Sie müssen zuerst das Paket distrib laden.
pdf_binomial (*k*, *n*, *p*) erstellt den Wert.

9.9 Binomialverteilung: bis k

Sie haben eine binomialverteilte Größe (n, p). Sie wollen die Wahrscheinlichkeit wissen, dass das Zufallselement kleiner gleich k mal vorkommt.

$$B(n,p)(X\leq k) = \sum_{i=0}^{k} \binom{n}{k}\, p^k\, q^{n-k}$$

```
(%i1) load (distrib)$
(%i2) cdf_binomial (3, 5, 0.2);
(%o2)                        .9932799999999998
(%i3) cdf_binomial (5, 5, 0.2);
(%o3)                                1
```

Abbildung 9.11: Der Code ist hier: 323

Sie müssen zuerst das Paket distrib laden.
cdf_binomial (*k*, *n*, *p*) erstellt die Summe. **cdf_binomial** (*n*, *n*, *p*) ist die Summe bis n und ergibt dann 1.

9.10 Permutation – Anreihung

Sie haben drei Personen Willi, Erna und Paul und möchten alle Möglichkeiten der Anreihung aufgelistet bekommen.

```
(%i1) teilnehmer : [Willi, Erna, Paul];
(%o1)                          [Willi, Erna, Paul]
(%i2) permut (teilnehmer);
resolvante
generale

NOTE: To compile the system do
load("sym/compile");
WARNING: DEFUN/DEFMACRO: redefining macro LGI in
         /usr/share/maxima/5.22.1/share/sym/util.lisp, was defined in
         /usr/share/maxima/5.22.1/share/sym/macros.lisp
\vdots
(%o2) [[Willi, Erna, Paul], [Willi, Paul, Erna], [Erna, Willi, Paul],
                  [Paul, Willi, Erna], [Erna, Paul, Willi],
                  [Paul, Erna, Willi]]
```

Abbildung 9.12: Der Code ist hier: 324

Der Befehl **permut** (*Liste*) gibt die Permutationen aus.

9.11 Regressionsgerade zu Daten erstellen

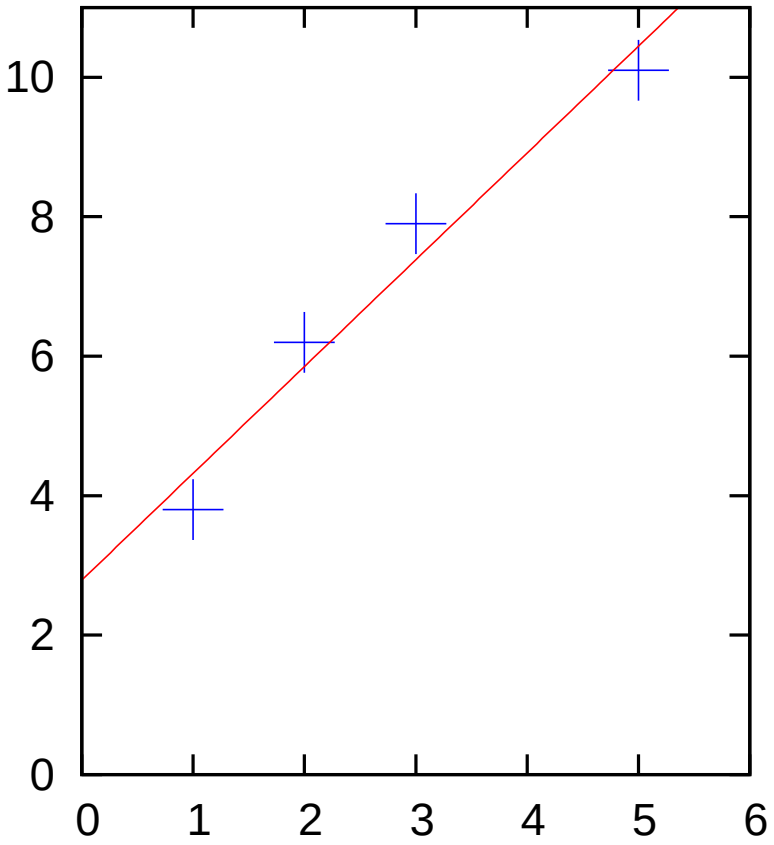
Sie haben Daten: entweder in einer Datei abgespeichert oder in einer Liste eingetippt. Nun wollen Sie eine Ausgleichsgerade erstellen lassen und die Daten visualisieren.

In der Datei „statistikregression.data“ sind folgende Daten:

1 3.8
2 6.2
3 7.9
5 10.1

```
(%i1) load (stats)$
(%i2) daten : read_nested_list (file_search ("statistikregression.data"));
(%o2)              [[1, 3.8], [2, 6.2], [3, 7.9], [5, 10.1]]
(%i3) xmax : lmax ( map (lambda ([x], x[1]), daten) );
(%o3)                                  5
(%i4) ymax : lmax ( map (lambda ([x], x[2]), daten) );
(%o4)                                 10.1
(%i5) z : simple_linear_regression (daten);
             |               SIMPLE LINEAR REGRESSION
             |
             |  model = 1.531428571428572 x + 2.788571428571426
             |
             |          correlation = .9815467064562255
             |
             |          v_estimation = .3894285714285719
             |
(%o5)        | b_conf_int = [.6237205475509736, 2.43913659530617]
             |
             |         hypotheses = H0: b = 0 ,H1: b # 0
             |
             |           statistic = 7.259168311737677
             |
             |           distribution = [student_t, 2]
             |
             |           p_value = .005393970462816311
(%i6) g : take_inference (model, z);
(%o6)               1.531428571428572 x + 2.788571428571426
(%i7) ausgleichsgerade : [
    color = red,
    explicit (g, x, 0, 6)
    ]$
(%i8) punkte : [
    point_size = 3,
    points (daten)
    ]$
(%i9) set_draw_defaults (
    dimensions = [500, 500],
    xrange = [0, 6],
    yrange = [0, 11]
    )$
(%i10) draw2d (
    file_name = "regressionsgerade",
    terminal = jpg,
    punkte, ausgleichsgerade
    )$
```

Abbildung 9.13: Der Code ist hier: 325



load (*stats*) lädt das Statistikpaket so dass Sie **simple_linear_regression** (*x*) zur Verfügung haben. x ist entweder eine 2-spaltige Matrix oder eine Liste, deren Elemente jeweils eine 2-elementige Liste sind.

file_search (*Dateiname*) gibt den Dateipfad zurück.

read_nested_list (*Datei*) liest die Datei und gibt eine verknüpfte Liste zurück. Diese Liste hat so viele Elemente, wie die Datei Zeilen hat. Jedes der Elemente ist aber wiederum eine Liste und besteht aus den einzelnen Wörtern (durch Leerzeichen abgetrennte Zeichenketten). Das Trennzeichen können Sie frei einstellen: **read_nested_list** (*Datei, separator_flag*) **separator_flag** kann sein: comma (Komma), pipe (|), semicolon (Semikolon) und space (Leerzeichen, default).

Zuerst werden der maximale x-Wert und der maximale y-Wert ermittelt. Vergleichen Sie auch Kap. ListeMinimumkap, S. 192. **map** (*Funktion, Liste*) erstellt eine Liste von x-Werten, die dann an **lmax** (*Liste*) übergeben wird. In der darauffolgenden Zeile erstellt **map** (*Funktion, Liste*) erstellt eine Liste von y-Werten. Dazu wird jeweils die anonyme Funktion **lambda** (*Variablenliste, Anweisungen*) benutzt. Jedes Element von daten wird an lambda übergeben. x ist eine Liste, die sich aus den Zeilen zusammensetzt. Von dieser Liste (Zeile) wird jeweils das erste bzw. zweite „Wort“ zurückgegeben.

simple_linear_regression (*x*) führt die Regression durch. Es wird ein sogenanntes Objekt zurückgegeben. Wenn Sie einzelne Elemente dieses Objektes haben wollen, wie z. B. die Regressionsgerade, welche unter model abgespeichert ist, dann geht das mit **take_inference** (*()*name, Objekt). Bei name geben Sie den Namen des Sie interessierenden Wertes ein.

Dann werden zwei Bilder erstellt. Eins für die Ausgleichsgerade und ein Bild für die Datenpunkt.

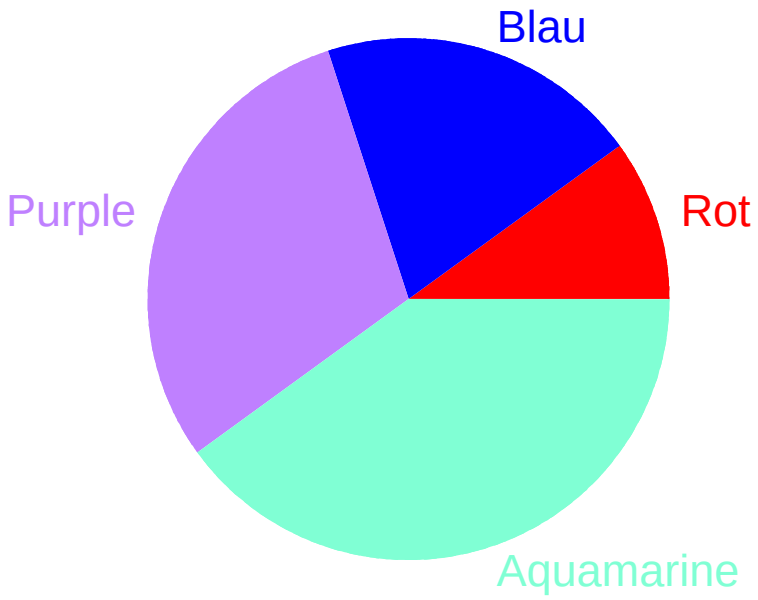
9.12 Kreisdiagramm erstellen

Sie wollen ein Kreisdiagramm erstellen. Die Daten haben Sie in einer Liste gespeichert, deren Elemente 2-elementige Listen sind. Die Elemente bestehen aus der Anzahl und einer Benennung. Sie haben also 10 Perso-

nen, welche rot wählen, 20, welche blau, 30, welche purple und 40 welche aquamarin gewählt haben.

```
(%i1) load (draw)$
(%i2) kreisdiagramm (L) := block ([i],
  gesamt: apply ("+", map (lambda ([x], x[1]), L)),
  w : append ([0], map (lambda ([x], x[1] / gesamt * 360), L)),
  farbnamen : [red, blue, purple, aquamarine, brown, yellow, dark-orange,
               green, magenta, black, light-goldenrod, grey],
  tmpliste : [],
  winkel_summe : 0,
  for i : 1 while i < length (w) do
    block ([],
      winkel_summe : winkel_summe + w[i],
      alpha : winkel_summe + w[i+1] / 2,
      x : 1.1 * cos (alpha / 360 * 2 * %pi),
      y : 1.1 * sin (alpha / 360 * 2 * %pi),
      if (x > 0) then
        la : left
      else
        la : right,
      tmpliste : append (tmpliste,
        [
          color = farbnamen [i],
          fill_color = farbnamen [i],
          label_alignment = la,
          label ([string (L[i][2]), x, y]),
          ellipse (0, 0, 1, 1, winkel_summe, w[i+1])
        ]
      )
    ),
    return (tmpliste)
)$
(%i3) liste : [[10, Rot], [20, Blau], [30, Purple], [40, Aquamarine] ]$
(%i4) segmentenliste : kreisdiagramm (liste)$
(%i5) draw2d (
    file_name = "kreisdiagramm",
    terminal = jpg,
    dimensions = [500, 500],
    user_preamble = ["set noborder"],
    xtics = false, ytics = false,
    proportional_axes = xy,
    xrange = [-1.5, 1.5],
    yrange = [-1.5, 1.5],
    nticks = 200,
    segmentenliste
    )$
```

Abbildung 9.14: Der Code ist hier: 327



Die Funktion kreisdiagramm ist etwas trickreich. In der Funktion werden zwei Dinge gleichzeitig gemacht: Es werden Beschriftungen (Label) erstellt und die Kreissegmente werden mit Hilfe von ellipse erstellt.

Die Funktion kreisdiagramm () erstellt eine Liste für draw2d für jedes einzelne Segment. Für das erste Segment sieht der Eintrag so aus:

- color = red,
- fill_color = red,
- label_alignment = left,
- label ([Rot, $1.1 \cdot \cos(\frac{pi}{10})$, $\sin(\frac{pi}{10})$]),
- ellipse(0, 0, 1, 1, 0, 36),

In dieser Lösung werden die Kreissegmente für das Kreisdiagramm mit Hilfe von **ellipse** (*Mx, My, a, b, Winkelstart,* W gezeichnet. Die beiden Halbachsen a und b werden als 1 gewählt, so dass ein Kreis entsteht. Der Mittelpunkt wird mit Mx = 0 und My = 0 in den Ursprung gelegt. Winkelstart ist dann der Winkel ab wo das Segment gezeichnet wird und Winkelsegment ist der Winkel des Kreisausschnittes.

Der wesentliche Bestandteil der Funktion ist die Liste w, welche die Winkel jedes einzelnen Segmentes enthält. Mit Hilfe dieser Liste werden die Positionen der Beschriftungen (Label) und die Kreissegmente bestimmt.

Zuerst wird mit **load** (*draw*) das Paket geladen, welches die Funktion draw, bzw. draw2d erlaubt.

Der Funktion kreisdiagramm wird eine Liste übergeben, deren Elemente wiederum 2-elementige Listen sind, bestehend aus der absoluten Anzahl und der Beschriftung.

Da Sie jetzt viele Anweisungen haben, werden diese in einem block zusammengefasst. i ist dabei die lokale Variable, die ausserhalb des Blockes nicht bekannt ist.

Nun wird die Gesamtanzahl (gesamt) bestimmt. **map** (*Funktion, Liste*) durchläuft die Liste, wobei die Funktion **lambda** (*lokale Variablenliste, Anweisungen*) jeweils das erste Element jedes Elements von L (also immer die Anzahl) zurückgibt. **apply** (*Funktion, Liste*) summiert die Listenelemente auf.

In der Liste w werden jetzt die Winkel in Grad jedes einzelnen Elementes gespeichert. Der erste Winkel wird mit 0 festgelegt und mit der Liste der einzelnen Segmentswinkel mit Hilfe von **append** (*Liste, Liste*) zusammengefügt:
w : [0, 36, 72, 108, 144]
Insgesamt ergibt die Summe der Winkel $360°$. Da an erster Stelle die Null geschrieben ist, so haben sich alle Winkelpositionen verschoben: Der Winkel des ersten Kreissegmentes steht an der zweiten Stelle, der Winkel des zweiten Segmentes steht an der dritten Stelle usw.

In der Liste farbnamen werden die Reihenfolge der zu nutzenden Farbnamen festgelegt. Diese Liste muss mindestens so lang sein, wie die Anzahl der Kreissegmente. Es erfolgt in der Funktion kein Test darauf, ob die Länge der Listen (der Daten und der Farbnamen) kompatibel ist. Das muss bei dieser Funktion der Benutzer leisten.

tmpliste ist die Liste, welche später zurückgegeben wird mit return. Darin wird jetzt alles gespeichert. Damit Maxima weiß, dass tmpliste eine Liste sein soll, wird tmpliste als Liste deklariert durch:
tmpliste : [],

In der Variablen winkel_summe wird der Anfang des Segmentes in Grad gespeichert. Zu Beginn ist winkel_summe null.

Nun wird eine **for**-Schleife eingerichtet. Bei jedem Schleifendurchgang wird der Code für ein Kreissegment für draw2d erzeugt.

In winkel_summe wird beim ersten Durchgang 0 dazuaddiert. Beim nächsten Durchgang wird der Winkel des ersten Segmentes aufaddiert. So ist in der Variablen winkel_summe immer der Beginn des Kreissegmentes gespeichert.

alpha ist eine Variable, mit der der Ort der Beschriftung angegeben wird. alpha ist so gewählt, dass die Beschriftung in der Mitte des Segmentes angebracht wird.

In den Variablen x, bzw. y ist der x-Wert bzw. der y-Wert der Beschriftung angegeben. Da **cos** in radian berechnet wird (d. h. eine volle Umdrehung sind 2π) muss der Winkel von Grad in radian umgerechnet werden.

Wenn die Beschriftung rechts vom Kreis angebracht werden soll (x ist also größer als null), dann soll die angegebene Position (x, y) für die linke Ecke der Beschriftung gelten. Wenn die x-Position links vom Kreis sein soll, so dass x kleiner als null ist, so soll die Beschriftungsposisiton für die rechte Kante gelten. So reicht die Beschriftung nicht in den Kreis hinein.

Zu tmpliste wird jetzt der Code für ein Kreissegment angehängt. Da ellipse ein Kreissegment um den Ursprung zeichnen soll, muss der erste Teil der Argumente (0, 0, 1, 1, ...) sein. Der Beginn des Segmentes ist in winkel_summe gespeichert. Der Winkel des ersten Kreissegmentes ist in w[2] gespeichert, da das erste Element eine Null ist.

In draw2d ist mit **user_preamble**, **xtics = false** und **ytics = false** der Rahmen um das Bild entfernt.

Durch die Angabe **proportional_axes = xy** wird sichergestellt, dass die Achsen immer den gleichen Maßstab haben. Dies ist hier nicht nötig, weil **xrange** und **yrange** identisch sind..

Kapitel 10

Eingabe

In diesem Kapitel werden Möglichkeiten der Eingabe von Daten aus Dateien beschrieben.

10.1 Einlesen einer Liste aus einer Datei

Die Werte der Liste stehen in einer Datei. Nach jedem Listenwert erfolgt ein Zeilenumbruch oder ein Leezeichen.

In der Datei „listen.data“ steht folgendes:

1 2 3
4 5 6

```
(%i1) a : read_list (file_search ("listen.data"))$
(%i2) a;
(%o2)                          [1, 2, 3, 4, 5, 6]
(%i3) length (a);
(%o3)                                  6
(%i4) b : read_nested_list (file_search ("listen.data"))$
(%i5) b;
(%o5)                        [[1, 2, 3], [4, 5, 6]]
```

Abbildung 10.1: Der Code ist hier: 328

Sie können die Liste einlesen mit **read_list** (*Quelle, Array*) bzw. **read_list** (*Quelle, Array, Trennzeichen*). **file_search** (*Dateiname*) findet die Datei.

length (Liste) gibt Ihnen die Länge der Liste zurück. Das ist praktisch bei vielen Zahlen.

Sollten die einzelnen Werte der Matrix in der Datei durch ein spezielles Zeichen (z. B. ein Komma) getrennt sein, so können Sie das Trennzeichen angeben: **read_list** (*file_search (“listen.data“), comma*); Trennzeichen können sein: „comma“, „semicolon“, „space“ oder „pipe“ für |.

Wenn Sie jede Zeile als Liste schreiben lassen wollen, dann geht dies mit **nested_list** (). Da dann jede Zeile eine eigene Liste ist, müssen die Listen nicht alle gleich lang sein.

10.2 Einlesen einer Matrix

Sie haben in einer Datei die Daten einer Matrix

Diese Daten lesen Sie ein mit:

- **read_matrix** (*Quelle*)
  Dies gibt eine Matrix zurück:
  M : **read_matrix** (*“meindatensatz.asc“*)
- **read_matrix** (*Quelle, Matrix*)
- **read_matrix** (*Quelle, Trennzeichen*)
- **read_matrix** (*Quelle, Matrix Trennzeichen*)

```
(%i1) a : read_matrix (file_search ("matrix.data"));
                                    [ 1  2  4 ]
(%o1)                               [         ]
                                    [ 3  4  5 ]
(%i2) length (a);
(%o2)                                  2
(%i3) a;
                                    [ 1  2  4 ]
(%o3)                               [         ]
                                    [ 3  4  5 ]
```

Abbildung 10.2: Der Code ist hier: 329

length (*Matrix*) gibt Ihnen die Spaltenzahl an.

Sollten die einzelnen Werte der Matrix in der Datei durch ein spezielles Zeichen (z. B. ein Komma) getrennt sein, so können Sie das Trennzeichen angeben: **read_matrix** (*file_search (“matrix.data“), comma*); Trennzeichen können sein: „comma“, „semicolon“, „space“ oder „pipe“ für |.

10.3 Einlesen von Datensätzen oder einer Matrix in einen Array

Sie haben in einer Datei eine Matrix oder Daten, die jeweils zusammengehören:
Schmidt, Alter Weg, 100
Meier, Jägerstr., 12

Diese Daten lesen Sie mit **read_array** (*Quelle, Array*), bzw **read_array** (*Quelle, Array, Trennzeichen*) ein:

```
(%i1) array (A, 2, 3);
(%o1)                                  A
(%i2) read_array (file_search ("array.data"), A, comma);
(%o2)                                 done
(%i3) arraylist (A);
(%o3)                          arraylist(A)
(%i4) A;
(%o4)                                  A
(%i5) read_hashed_array (file_search ("array.data"), B, comma);
(%o5)                                  B
(%i6) arraylist (B);
(%o6)                          arraylist(B)
(%i7) B[Schmidt];
(%o7)                      [[Alter, Weg], 100]
```

Abbildung 10.3: Der Code ist hier: 330

Wenn Sie **read_array**() verwenden, dann bedenken Sie, dass Sie

1. das Array vorher definieren müssen
2. nicht A : **read_array** schreiben dürfen, denn sonst wird nur „done“ abgespeichert.

Bei einem hashed_array, ist der 1. Wert jeweils der Schlüssel, mit dem Sie die Zeile ansprechen können.

Sollten die einzelnen Werte der Matrix in der Datei durch ein spezielles Zeichen (z. B. ein Komma) getrennt sein, so können Sie das Trennzeichen angeben. Z.B: **read_array**(*file_search (“array.data“), comma;*) Trennzeichen können sein: „comma“, „semicolon“, „space“ oder „pipe“ für |.

Kapitel 11

Polynome

11.1 Multiplikation von Polynomen

Sie wollen zwei Polynome multiplizieren:

```
(%i1) (x-1)*(x+2) * (x^2 + 1);
                                          2
(%o1)                    (x - 1) (x + 2) (x  + 1)
(%i2) (x-1)*(x+2) * (x^2 + 1), expand;
                             4    3    2
(%o2)                       x  + x  - x  + x - 2
```

Abbildung 11.1: Der Code ist hier: 331

Durch expand wird der Ausdruck ausmultipliziert.

11.2 Faktorisieren

Sie wollen eine Polynom faktorisieren:

```
(%i1) p1 : x^4 + x^3 - x^2 + x - 2;
                             4    3    2
(%o1)                       x  + x  - x  + x - 2
(%i2) factor (p1);
                                          2
(%o2)                    (x - 1) (x + 2) (x  + 1)
(%i3) algsys ([p1], [x]), realonly: true;
(%o3)                       [[x = 1], [x = - 2]]
```

Abbildung 11.2: Der Code ist hier: 332

factor gibt Ihnen eine Faktorisierung mit reellen Zahlen. Die Faktoren sind kleinstmöglich gewählt.

Wenn Sie die rellen Nullstellen haben wollen, empfiehlt sich **algsys** (*Liste der Polynome, Liste der Variablen*) mit der Option **realonly** : true. Diese Funktion gibt Ihnen die gemeinsamen Nullstellen der Polynome in einer ansehnlichen Form. Standardmäßig ist die Option **realonly** : false gesetzt.

11.3 Polynomdivision

Sie wollen zwei Polynome dividieren:

```
(%i1) p1 : x^2 + x + 1;
                                    2
(%o1)                              x  + x + 1
(%i2) p2 : x;
                                     x
(%o2)
(%i3) p3 : x^2;
                                     2
(%o3)                               x
(%i4) p1 / p2;    /* Schreibt den Bruch */
                                    2
                                   x  + x + 1
(%o4)                              ----------
                                      x
(%i5) ratsimp(%); /* Vereinfacht den vorherigen Ausdruck */
                                    2
                                   x  + x + 1
(%o5)                              ----------
                                      x
(%i6) divide (p1, p2, x);
(%o6)                              [x + 1, 1]
(%i7) divide (p1, p3, x);
(%o7)                              [1, x + 1]
```

Abbildung 11.3: Der Code ist hier: 333

Benutzen Sie das geteilt Zeichen oder **divide** (*Polynom1, Polynom2, Variablenliste*). **divide** (. . .) gibt Ihnen eine Liste mit zwei Elementen zurück:

1. Das erste Element ist das Ergebnis der Division.
2. Das zweite Element ist der Rest, welcher bei der Division entsteht.

11.4 Asymptote finden

Sie haben eine gebrochen rationale Funktion und wollen die Asymptote derselben finden. Dann haben Sie eine Division mit Rest.

```
(%i1) f(x) := (x^2 + 2*x ) / ( x + 1 );
                                          2
                                         x  + 2 x
(%o1)                          f(x) := --------
                                          x + 1
(%i2) partfrac (f(x), x);
                                      1
(%o2)                          - ----- + x + 1
                                    x + 1
```

Abbildung 11.4: Der Code ist hier: 334

Die Asymptote von $f(x)$ ist $g(x) = x + 1$.

Kapitel 12

Eigene Funktionen / Programme

In diesem Kapitel werden Ihnen Beispiele vorgestellt, wie Sie eigene Funktionen schreiben können. Dies ist der Übergang zur Programmierung.

12.1 Benutzung eines Blockes

Wenn Sie mehrer Anweisungen zusammenfassen wollen, dann müssen Sie eine eigene „Umgebung“ schaffen. Dies machen Sie bei maxima mit einem **block** (*[lokale Variablen], Anweisungen*).

Anweisungen zusammmenfassen müssen Sie, wenn Sie eine Funktion schreiben. Maxima muss ja schliesslich wissen, wann die Funktion zu Ende ist. Auch bei for-Schleifen oder bei einer Bedingung (**if**) die ja immer nur auf die nächste Anweisung wirken benötigen Sie gegebenenfalls einen Block. Wenn nach einer Bedingung mehrere Anweisungen durchgeführt werden sollen, dann müssen Sie diese alle zusammenfassen.

Sie können lokale Variablen angeben, die nur innerhalb dieses Blockes gültig sind und auf die später nicht mehr zugegriffen werden kann und die andererseits nicht schon vorhandene Variablen verändern.

Stellen Sie sich vor, dass Sie eine längere Datei mit Anweisungen für maxima geschrieben haben. Sie haben im gesamten Text mit der Variable Laenge gearbeitet. Dann kommt eine Funktion in der Sie auch eine Länge berechnen. Dann wollen Sie nicht, dass diese Variable den Rest Ihres Textes beeinflußt.

Wenn Sie keine lokalen Variablen haben, dann können Sie auch nur mit runden Klammern einen Block erzeugen.

Innerhalb eines Blockes schliessen Sie einzelne Anweisungen mit einem Komma ab.

Der Wert eines Blockes kann durch return (Rückgabewert) gesetzt werden oder es ist einfach der letzte Wert des letzten Ausdrucks.

```
(%i1) x : 3$            /* Diese Variable ist global */
(%i2) if x > 2 then
(%i2)   (
(%i2)     print ("x ist", x),
(%i2)     x : 5         /* x ist global */
(%i2)   )$
x ist 3
(%i3) x;
(%o3)                                5
(%i4) if x > 2 then
  block ([x : 7],
    x : x + 1,    /* x ist nur lokal gueltig */
    print ("x ist", x)
  )$
x ist 8
(%i5) x;
(%o5)                                5
```

Abbildung 12.1: Der Code ist hier: 335

x erhält den Wert 3. x nennt man eine globale Variable.

In dem Block ist x nicht als lokal definiert, also wird auf die globale Variable zugegriffen (Die Variable der nächsthöheren Umgebung wird benutzt).

Dementsprechend hat dann die Variable x nach dem Block den Wert 5.

Im nächsten Block ist x lokal definiert. Sie können der lokalen Variablen sogar einen Wert zuweisen. Diese Variable können Sie innerhalb des Blockes verändern ohne dass die globale Variable mit dem gleichen Namen davon beeinflusst wird.

Dementsprechend hat dann die Variable x nach dem Block den alten Wert 5 behalten.

Wenn Sie sich selbst Funktionen schreiben, die Sie immer mal wieder benutzen wollen, dann ist es sehr wichtig, dass Sie alle Variablen sauber als lokal definieren. Dann müssen Sie nicht erst nachschauen, ob derselbe Variablenname in Ihrem Text und in der Funktion vorkommt.

In einer Funktion auf globale Variablen zuzugreifen mag für kleinere Dateien, welche überschaubar sind, bequem sein. Dennoch sollten Sie sich klar sein, dass dies ein sehr gefährlicher und schlecht zu wartender Programmcode wird und dementsprechend eigentlich nicht zur Anwendung kommen.

12.2 Länge eines Vektors als Programmbeispiel

Gesucht ist eine Funktion, welche die Länge eines übergebenen Vektors zurückgibt. Dieses Beispiel soll Ihnen die Funktionsprogrammierung näher bringen. Eine sinnvolle und schnelle Lösung finden Sie im Paket vector_rebuild mit der Betragsfunktion $|a|$ (siehe auch: Kap. 8.6, S. 85). Für dieses Problem gibt es selbstredend auch noch andere Lösungsansätze. Dies soll ein Beispiel sein für eine for-Schleife.‘

```
(%i1) len (vec) :=  block ([sum : 0],
    for n in flatten ( args(vec) ) do
      sum : sum + n^2,
    sqrt(sum)
  )$
(%i2) load (eigen)$
(%i3) a : covect ([3, 4]);
                                        [ 3 ]
(%o3)                                   [   ]
                                        [ 4 ]
(%i4) len (a);
(%o4)                                    5
(%i5) len ([3, 4]);
(%o5)                                    5
(%i6) load (vector_rebuild)$
(%i7) |a|;  /* Funktion in vector_rebuild */
(%o7)                                    5
```

Abbildung 12.2: Der Code ist hier: 336

len (vec) benennt die Funktion mit dem Namen len und zeigt an, dass vec übergeben wird.

Durch **block** (*[lokale Variablenliste], Befehle*) definieren Sie einen Block. [sum] ist dann eine lokale Variable innerhalb des Blockes.

sum wird mit dem Wert null belegt.

flatten (*Ausdruck*) erzeugt eine einzige Lister:
flatten ([3, [4,5], 6);
ergibt folgende Liste: [3, 4, 5, 6]
Sollte der vec aus Listen von Listen bestehen, so werden alle Elemente aus den verschiedenen Listen in eine einzige Liste geschrieben.

args (*Ausdruck*) gibt eine Liste der Argumente von der Variablen vec. Aus einem columnvector werden die Werte - Argumente „herausgezogen“ und in eine Liste gesteckt. Daher kann hier vec ein Spaltenvektor sein.

Dann wird eine **for** Schleife erzeugt. In der nimmt n alle Werte aus der von **flatten** erstellen Liste an.

Die Variable sum wird bei jedem Durchgang um den Wert n^2 erhöht. Dadurch wird die Summe der Quadrate der Elemente gebildet.

Der Block wird mit der Berechnung der Wurzel von sum beendet. Der Wert des Blockes ist der Wert des letzten Ausdrucks also der Wurzel von sum.

Das Paket eigen stellt Ihnen **covect** (*Liste*) zur Verfügung.

Die Funktion können Sie dann mit **len (a)** aufrufen.

Das Paket vector_rebuild stellt Ihnen ebenfalls **covect** (*Liste*) aber zusätzlich auch die Betragsfunktion zur Verfügung. zur Verfügung.

12.3 Variablensubstitution einer Lösung von solve

Bei **solve** (*Gleichungen, Variablenliste*) werden Ihnen bei mehreren Lösungen die Parameter als %r1 usw. angegeben. Dies wollen Sie für maximal drei Lösungsparameter ändern.

```
(%i1) mysolve (a, b) := block ([lsg, varListe, len],
    lsg : solve (a, b),
    varListe : [r, s, t],
    len : min (3, length (%rnum_list) ),
    for i thru len do
      lsg : subst (varListe[i], %rnum_list[i], lsg),
    lsg
  )$
(%i2) gl1 : 2 * x = y;
(%o2)                                2 x = y
(%i3) gl2 : 2 * gl1;
(%o3)                               4 x = 2 y
(%i4) gleichungen : [gl1, gl2];
(%o4)                        [2 x = y, 4 x = 2 y]
(%i5) variablen : [x, y];
(%o5)                                [x, y]
(%i6) mysolve (gleichungen, variablen);
solve: dependent equations eliminated: (2)
                                      r
(%o6)                          [[x = -, y = r]]
                                      2
(%i7)
/* Alternative ohne for */
mysolve (a, b) := block ([lsg, varListe, maperror:false],
    lsg : solve (a, b),
    varListe : [r, s, t],
    lsg : subst (map ("=", %rnum_list, varListe), lsg)
  )$
(%i8) mysolve (gleichungen, variablen);
solve: dependent equations eliminated: (2)
map: truncating one or more arguments.                                  r
(%o8)                          [[x = -, y = r]]
                                      2
(%i9) %;
                                      r
(%o9)                          [[x = -, y = r]]
                                      2
```

Abbildung 12.3: Der Code ist hier: 337

Es werden Ihnen zwei Lösungsmöglichkeiten angeboten. Einmal mit einer for-Schleife und die zweite Möglichkeit benutzt **map** (*Funktion, Ausdruck* . . .). Die zweite Möglichkeit zeigt, dass es in vielen Fällen eine Alternative zu der in anderen Programmiersprache üblichen for-Schleife gibt indem Sie mit Listen arbeiten.

12.3.1 Alternative 1: for-Schleife

Es wid eine Funktion namens mysolve erstellt, welcher zwei Gleichungen übergeben wird in a und b.

Alle Anweisungen der Funktion werden in einen Block zusammengefasst, der die lokalen Variablen lsg, varListe und len enthält.

In lsg wird die Lösung von solve gespeichert. Diese enthalt als Variablen %r1 usw. Diese Variablen sind in %rnum_list gespeichert.

varListe ist eine Liste mit den neuen zu verwendenden Variablennamen.

length(%rnum_list) gibt die Anzahl der Elemente der Liste %rnum_list. In dieser Liste stehen die benutzen Variablennamen welche solve benutzt hat.

Sie können schließlich nicht auf das nicht existierende 4. Element von varListe zugreifen. Daher ist len das Minimum von 3 oder der Länge von %rnum_list.

for i thru len do
Diese Anweisung ist eine for-Schleife und ersetzt in der folgenden Anweisung das i durch 1, 2, . . . , len.

subst (*a, b, c*) ersetzt in der Zeichenkette c jedes Vorkommen des Ausdrucks b durch a. Hier wird also jeder Name aus %rnum_list in lsg durch den entsprechenden Namen aus varListe ersetzt.

12.3.2 Alternative 2: map

maperror : false verhindert einen Abbruch, wenn die Ausdrücke bei map nicht alle gleich lang sind.

lsg : subst (map (“=“, %rnum_list, varListe), lsg),
besteht aus einem inneren Teil: **map** und einem äußeren Teil.

map (*Funktion, Ausdruck, Ausdruck, . . .*) wendet die Funktion auf jeden einzelnen Ausdruck an. Z. B. erzeugt: map (“=“, [%r1, %r2], [r, s, t]) folgende Liste: [%r1 = r, %r2 = s]

Hier ersetzt **subst** (*varListe[i], %rnum_list[i], lsg*) in der Zeichenkette lsg die Buchstabenkokmbination subst ([a = b], expr) entspricht subst (b, a, expr)

Leider erzeugt maperror : false eine Fehlermeldung ohne Zeilenumbruch. Deshalb müssen Sie mit die %; die Ausgabe nochmal erfolgen lassen.

12.4 Zufallswerte

Sie benötigen eine Liste von 10 Zufallswerten zwischen 1 und 6 z. B. zur Simulation eines Würfels.

```
(%i1) liste : makelist (random(5) + 1, i, 1, 10);
(%o1)                  [3, 3, 5, 1, 5, 2, 5, 1, 4, 4]
```

Abbildung 12.4: Der Code ist hier: 338

random (*x*) gibt Ihnen eine Zufallszahl zurück.

- Wenn x eine natürliche Zahl ist, dann ist die zurückgegebene Zufallszahl zwischen 0 und x-1 (jeweils eingeschlossen).
- Wenn x eine positive Dezimalzahl ist, dann erhalten Sie eine zufällige Dezimalzahlqahl zwischen 0 und 1 jeweils inkl.

Wenn Sie also einen Wert zwischen 1 und 6 haben wollen, dann erstellt Ihnen **random(5)** Werte zwischen 0 und 5 (einschliesslich). Erhöhen Sie diese Wert um eins und dann haben Sie Werte zwischen 1 und 6.

makeliste (*Ausdruck, Variable, kleinster Wert der Variable, größter Wert der Variable*) erstellt eine Liste, bei der die Variable immer um eins erhöht wird und in den Ausdruck eingesetzt wird. Hier benötigen wir die Variable nicht. Denn bei allen Listenelementen soll unabhängig von der Position des Elements immer nur ein

Zufallswert zwischen 1 und 6 gebildet werden. Aber makelist gibt uns eine einfache Möglichkeit 10 Listenelemente zu bilden.

12.5 Bedingungen

Sie wollen eine Bedingung überprüfen lassen. Z. B. ob ein Wert größer ist als 5

```
(%i1) f(x) :=
  (
    if x > 2 then
      (
        print (x, " ist groesser als 2"),
        print ("Geben Sie eine andere Zahl ein")
      )
    else
      print (x, " ist kleiner als 2")
  )$
(%i2)
f(3)$
3  ist groesser als 2
Geben Sie eine andere Zahl ein
(%i3) f(1)$
1  ist kleiner als 2
(%i4) f(1);
1  ist kleiner als 2
(%o4)                          ist kleiner als 2
```

Abbildung 12.5: Der Code ist hier: 339

Bedingungen werden mit **if** (*Ausdruck*) überprüft. Die Struktur sieht folgendermaßen aus:

1. if *Bedingung* then
   nun folgt **eine** Anweisung!
   Wenn Sie mehr als eine Anweisung haben wollen müssen Sie diese Anweisungen zusammenfassen.

2. if *Bedingung* then
   nun folgt eine Anweisung!
   Wenn Sie mehr als eine Anweisung haben wollen müssen Sie diese Anweisungen zusammenfassen.
   else
   nun folgt wiederum eine Anweisung!
   Wenn Sie mehr als eine Anweisung haben wollen müssen Sie diese Anweisungen zusammenfassen.

Sie können Anweisungen durch runde Klammern zusammenfassen oder durch:
block ([lokale Variablen], . . .) Ausserhalb des Blockes können Sie auf die lokalen Variablen nicht zugreifen.

Der letzte Aufruf: f(x); erfordert durch das Semikolon eine Ausgabe. Der Wert des Blockes ist sein letzter Ausdruck und dies ist „ist kleiner als 2“.

12.6 Erreichen der Grenzverteilung einer stochastischen Matrix

Sie haben eine stochastische Matrix und ein Dreieck und wollen mit der Matrix die Eckpunkte des Dreiecks verändern und die Grenzverteilung sichtbar machen.

```
(%i1) load (draw)$
(%i2) M : matrix (
    [0.95, 0.05],
    [0.05, 0.95]
    )$
(%i3) /* Anzahl der Bilder */
n : 45$
(%i4) X : matrix ([x], [y])$
(%i5) my_gnuplot_preamble : [
    "set yrange [0:8]",
    "set xrange [0:8]"
    ]$
(%i6) for i : 1 while i < n do
  block ([],
    bild[i] : gr2d (
      title = concat ("potenz = ", string(i)),
      transform = [
            (M^^i . X)[1][1],
            (M^^i . X)[2][1],
            x, y
            ],
      triangle ( [3,2], [7,2], [5,5] )
    )
  )$
(%i7) bilder : [bild[1]]$
(%i8) for i : 2 while i < n do
  bilder : append (bilder, [bild[i]])$
(%i9) set_draw_defaults(
    user_preamble = my_gnuplot_preamble,
    dimensions = [500, 500],
    yrange = [0, 8],
    xrange = [0, 8],
    border = false
    )$
(%i10) draw (
    delay = 80,
    file_name = "stochastischerUebergang",
    terminal = 'animated_gif,
    bilder
    )$
End of animation sequence
```

Abbildung 12.6: Der Code ist hier: 340

Aus Platzgründen werden die Befehle mit einem Dollarzeichen abgeschlossen. Das ist nicht empfehlenswert im normalen Gebrauch.

Das Paket **load** (*draw*) wird geladen.

Die Matrix M wird definiert. Dies ist eine stochastische Matrix.

n gibt die Anzahl der zu erstellenden Bilder an.

Um Gleichungen für **transform** zu erstellen, wird X benötigt. In X steht „x“ und „y“.

my_gnuplot_preamble wird als user_preamble in gnuplot eingefügt und dies sind direkte gnuplot-Befehle. Diese sorgen dafür, dass das Bild jeweils von 0 bis 8 geht.

Jetzt kommt eine Schleife mit **for**. Die Laufvariable i beginnt bei 1. Die nachfolgende Anweisung wird hier solange durchlaufen, wie i kleiner als n also kleiner als 45 ist.

Da wird hier mehr als eine Anweisung haben, die jedesmal mit unterschiedlichen Werten für i durchlaufen wird, werden diese Anweisungen alle mit einem Block zusammengefasst.

block (*[lokale Variablen], Anweisungen*)

Es werden nun 44 Bilder mit gr2d definiert. Der Titel wird mit **concat** (*Zeichenkette1, Zeichenkette2*) zusammengefügt zu „Potenz = 1“, „Potenz = 2“ usw. **string** (*Ausdruck*) gibt den Wert für den Ausdruck als Zeichenkette zurück.

führt die geforderte Transformation durch. M^^i .X erstellt Ihnen gerade eine Matrix. Die Transformation für den x-Wert ist gerade das Element 1, 1. Die Transformation für den y-Wert ist gerade das Elemente 2, 1. Die Transformationsvariablen sind x und y.

triangle (*[x1, y1], [x2, y2], [x3, y3]*) ist eine Option von gr2d, welche ein Dreieck erzeugt.

Nun werden die Bilder zu einer großen Liste zusammengefügt. Diese Liste heisst dann bilder. Dazu wird zuerst bilder die Liste mit dem einem Element bild[1] zugewiesen. Damit weiss Maxima, dass bilder eine Liste ist. Dann werden die anderen Bilder: bild[i] dann an die Liste bilder angehaengt.

draw (*z*)eichnet die Szene. delay ist eine Option bei animierten gifs und gibt in 1/1000 Millisekunden den zeitlichen Abstand der Bilder an.

12.7 Listenelemente aus einer Liste auswählen

Sie haben eine Liste und wollen aus dieser Liste Elemente an einer bestimmten Stelle auswählen: meineliste ist die Liste mit allen Elementen, und indliste ist die Liste mit den Positionen, welche ausgewählt sind:

```
(%i1) meineliste : [a, b, 4, 5]$
(%i2) indliste : [1, 3]$
(%i3) makelist (meineliste[x], x, indliste);
(%o3)                                [a, 4]
```

Abbildung 12.7: Der Code ist hier: 341

Sie definieren Ihre Listen und erstellen mit **makelist** (*Ausdruck, x, Liste*) eine neue Liste. x durchläuft alle Elemente der Liste, die Sie makelist übergeben. Sie erhalten also ganau so viele Elemente wie Sie sie in Liste haben. In Ausdruck wird dann jeweils das x durch das jeweilige Element der Liste ersetzt.

Hier erhalten Sie also eine Liste aus zwei Elementen, weil indliste zwei Elemente hat.

indliste wird durchlaufen:

- 1. Element von indliste: 1, dadurch gilt: x = 1, meineliste[1] ist dann das erste Element der Liste, welche makeliste zurückgibt.
- 2. Element von indliste: 3, dadurch gilt: x = 3, meineliste[3] ist dann das zweite Element der Liste, welche makeliste zurückgibt.

(Siehe auch: Kapitel 6.7, S. 38).

Kapitel 13

Graphik

Es gibt in Maxima mehrere Möglichkeiten der Visualisierung.
Für Funktionen:

1. **plot2d** (. . .)
2. **draw** (*gr2d, . . . , gr3d, . . . , Optionen*)
   gr2d erstellt Maxima-Object, welches ein Bild oder eine Szene im zweidimensionalen beschreibt.
3. **draw2d** (*Optionen, Graphikobject*)
   Bei **draw2d** (. . .) übergeben Sie direkt die Optionen an gr2d. Dies ist eine Abkürzung für: **draw** (*gr2d(. . .)*).

Für Statistik:

1. **boxplot** (*Boxplot*)
2. **histogram** (*Histogramm*)
3. **barsplot** (*Balkendiagramm*)
4. **scatterplot** (*Streudiagramm*)

Um Netze bzw. Graphen zu zeichnen:

1. **draw_graph** (*graph*), bzw. **draw_graph** (*graph, Optionen, . . .)*

Diese Programme leiten die Information entweder an das Programm Gnuplot oder an Xmaxima. Gnuplot ist ein Programm, welches bei der Erstellung eines Bildes sehr viele Optionen bietet. Wenn Sie sich mit Gnuplot auskennen, können Sie alle Gnuplot-Befehle weiterreichen. Das ist mit den unterschiedlichen Plotfunktionen unterschiedlich. Dies ist demonstriert in Kap. 13.4 auf Seite 130.

1. **plot2d** (*b*)enutzt dazu einen String, in dem die Gnuplot-Befehle durch Semikolons abgetrennt sind und als Option übergeben werden:
   [gnuplot_preamble, gnuplotstr].
2. **draw2d** (*v*)ersteht sich als grnuplot-Frontend. Das heisst, Sie können sehr viele der Gnuplot-Optionen mit den Optionen von draw2d einstellen.

   Wenn Sie Steuerungsbefehle direkt an Gnuplot weiterreichen wollen, ist die Methode anders. Sie übergeben eine Liste, bei der die Befehle durch Kommata abgetrennt sind als Option:
   user_preamble = my_gnuplot_preamble,
   Da in der Liste die Zeilen eine Zeichenkette ergeben sollen, müssen Sie jede Zeile in Anführungszeichen schreiben. Wenn Sie ein Anführungszeichen bei dem Gnuplot-Befehl benötigen, dann benutzen Sie ein einfaches Hochkomma.

Die Unterschiede zwischen **plot2d**(...) und **draw2d**(...) gehen noch weiter, weil Sie mit **plot2d**(...) einen zu zeichnenden Graphen ganz normal eingeben, aber bei **draw2d**(...) diesen erst dem Befehl **explicit (f, x, xmin, xmax)** übergeben müssen.

draw2d(...) ist die Kurzform für **draw**(*gr2d(Ihre Optionen)*) Es gibt also einen Unterschied, ob Sie **draw**(...) oder **draw2d**(...) aufrufen. In einem Fall können Sie noch Optionen für draw setzen, im anderen Fall nicht.

Sie können die Größe des Bildes angeben. Dies ist sinvoll, damit die Schriftgröße der späteren Größe angepasst erscheint. Übergeben Sie der Funktion draw die Optionen: dimensions = [300, 300]
Wenn Sie das Bild in eine html-Datei einfügen wollen, ist diese Angabe besonders nützlich und führt dann zu besseren Ergebnissen als wenn Sie nur eine gif-Datei erzeugen und dann in html die Höhe und Breite der Darstellung angeben.

pic_width und **pic_height** gelten für gif, png, jpg und svg Dateien und gibt die Ausmaße des Bildes an. Diese Option ist aber veraltet und stattdessen nehmen Sie besser dimensions:
dimensions = [300, 300]

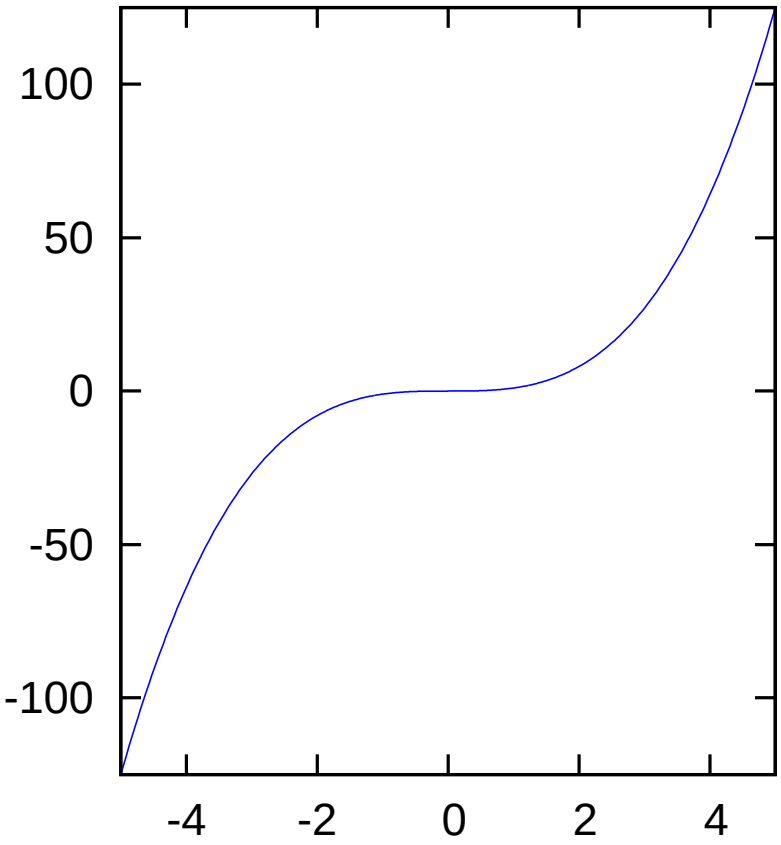
13.1 Zeichnen einer Funktion

Sie wollen die Funktion f zeichnen lassen.

$$f(x) = x^3$$

```
(%i1) load (draw)$
(%i2) f(x) := x^3;
                                        3
(%o2)                           f(x) := x
(%i3)
draw2d (
    file_name = "explizit",
    terminal = jpg,
    dimensions = [500, 500],
    explicit (f(x), x, -5, 5)
  )$
```

Abbildung 13.1: Der Code ist hier: 342



Um **draw** (*Graphikobjekte*) bzw **draw2d** (*Graphikanweisungen*) verwenden zu können müssen Sie zuerst das Paket mit **load** (*draw*) laden.

Der Übersicht halber wird zuerst eine Funktion definiert.

draw (*gr2d*) zeichnet ein Bild. Wenn Sie nur ein Bild zeichnen wollen, dann ist es einfacher **draw2d** (*Graphikanweisu*
zu verwenden.

Die einzelnen Graphikanweisungen werden durch Kommata getrennt.

Wenn Sie sich das Bild nur anschauen wollen, benötigen Sie weder **file_name** noch **terminal**. Dann wird das Bild auf Ihren Monitor gezeichnet. **file_name** gibt den Dateinamen ohne die Typbezeichunung an. **terminal** gibt den Typ an. Typen sind z. B.: screen, png, jpg, eps, pdf, gif, animated_gif, wxt.

Wenn Sie mehrere Bilder gleichzeitig in einem Fenster gezeichnet haben wollen, dann verwenden Sie mehrere gr2d() - Objekte in einem draw:

```
draw (
  gr2d ( ... ),
  gr2d ( ... )
)
```

In diesem Fall werden alle Ihre Bilder in ein Fenster gezeichnet.

dimensions wird angegeben, damit das jpg-Bild in der html-Ausgabe mit einer geeigneten Größe angegeben wird.

Sie können Funktionen explizit (explicit) oder implizit (implicit) verwenden. Die „normalen“ Funktionen in der Schule, bei denen auf einer Seite der Gleichung das y steht, sind alle explizit definiert.

Bei explicit geben Sie zuerst die Funktion an, dann wie die Variable heisst. Denn Maxima weiss nicht, ob die Variable der Funktion x oder z. B. t ist. Dann müssen Sie noch das Intervall angeben in dem die Variable gezeichnet werden soll.

Eine implizite Definition wäre z. B. folgende, welche einen Kreis angibt:

$$x^2 + y^2 = 1$$

Diese Zeichung ist eine recht minimalistische schnelle Darstellung einer Funktion. Sie haben hier jetzt kein „schönes“ Koordinatenkreuz. Die maximalen und minimalen y-Werte werden automatisch berechnet. Dies können Sie selbstverständlich mit geeigneten Optionen anpassen. In folgenden Beispielen wird oftmals mit Pfeilen ein Koordinatenkreuz gezeichnet.

13.2 Zeichnen einer impliziten Funktion

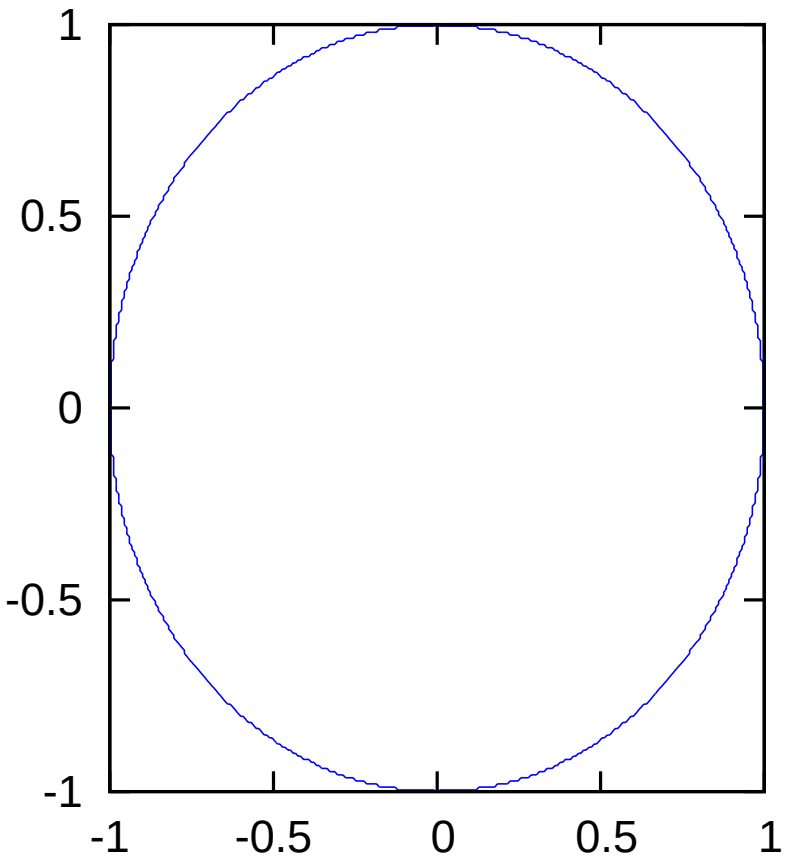
Sie haben eine implizite Funktion und möchten diese zeichnen lassen:

$$x^2 + y^2 = 1$$

Diese Gleichung beinhaltet zwei Funktionen (oberer Halbkreis und unterer Halbkreis).

```
(%i1) load (draw)$
(%i2) draw2d (
    file_name = "kreis",
    terminal = jpg,
    dimensions = [500, 500],
    implicit (x^2 + y^2 = 1, x, -1, 1, y, -1, 1)
  )$
```

Abbildung 13.2: Der Code ist hier: 343



Bei einer implizit definierten Funktion müssen Sie beide Variablen und deren Zeichenbereich angeben. Wenn y nur zwischen 0 und 1 gezeichnet wird, erhalten Sie z. B. einen oberen Halbkreis.

Bei einem Halbkreis empfiehlt es sich, den Zeichenbereich des Bildes mit Hilfe von xrange = [-1, 1] und vor allem yrange = [-1, 1] anzugeben. Dann wird Ihnen auch ein Halbkreis angezeichnet. Sonst wird das gezeichnetet Bild für den kleinsten und größten Wert automatisch angepasst.

13.3 Mehrere Fenster mit Zeichnungen

Sie möchten zwei Bilder auf Ihrem Monitor erzeugen und miteinander vergleichen.

```
load (draw)$
draw2d ( explicit ( x^2, x, -2, 2), terminal = [screen, 1])$
draw2d ( explicit ( x^3, x, -2, 2), terminal = [screen, 3])$
```

Abbildung 13.3: Der Code ist hier: 344

terminal = [screen, 1] Zeichnet ein Bild in Fenster Nr. 1. Wenn Sie screen = 2 angeben, wird in das (neue) Fenster Nr. 2 gezeichnet.

13.4 Schreiben in einem Graph

Sie wollen in einem Graph eine Markierung anbringen.

Dies können Sie entweder in Gnuplot schreiben oder als Option zu plot2d setzen.

Folgende Vorgehensweise empfiehlt sich:

1. Lassen Sie den Graph einmal zeichnen.
2. Lesen Sie die Koordinaten ab, an denen Ihre Markierung erscheinen soll. Die Koordinaten der Markierung sind relativ zu dem jeweiligen gewähltem Koordinatensystem.

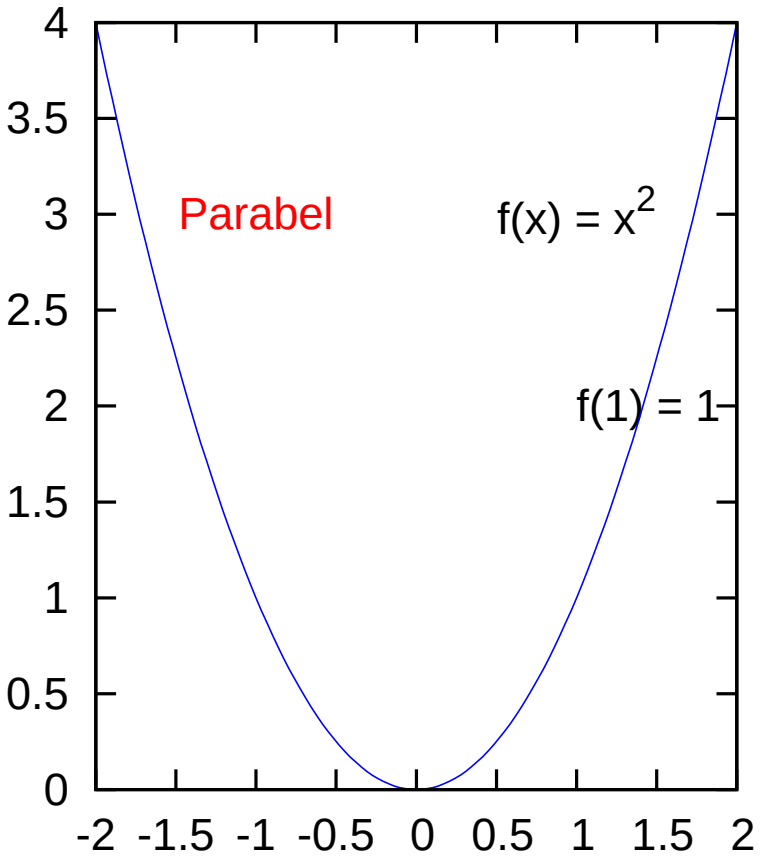
13.4.1 draw2d

In **draw2d**(*Optionen*, *Graphikobjekt*) können Sie Label (Markierungen) direkt angeben oder auch Gnuplot-Befehle einfach an Gnuplot übermitteln. Übergeben Sie Ihre Zeichenkette dann mit **user_preamble** = **...**

Bedenken Sie, dass **draw2d**(. . .) kein „jpeg“ kennt sondern nur „jpg“

```
(%i1) load (draw)$
(%i2) f(x) := x^2;
                                        2
(%o2)                            f(x) := x
(%i3)
str1 : concat ("set label 'f(1) = ", string ( f(1) ) );
(%o3)                          set label 'f(1) = 1
(%i4) str1 : concat (str1, "' at 1, 2;");
(%o4)                    set label 'f(1) = 1' at 1, 2;
(%i5)
my_gnuplot_preamble : [
    str1,
    "set label 'f(x) = x^2' at 1, 3 center"
    ]$
(%i6)
draw2d (explicit (f(x), x, -2, 2),
        file_name = "graphiklabel1",
        terminal = jpg,
        dimensions = [500, 500],
        user_preamble = my_gnuplot_preamble,
        color = red,
        label (["Parabel", -1, 3])
        )$
```

Abbildung 13.4: Der Code ist hier: 345



13.4.2 plot2d mit Gnuplot Befehlen manipulieren

Der Gnuplot-Befehl zum setzen einer Markierung bei den Koordinaten 1, 2 lautet im einfachsten Fall:
set label “Text“ at 1, 2
Sie können den Text mit einem frei wählbaren Winkel drehen lassen, die Fonts neu auswählen, bzw. vergrößern, die Farbe einstellen und sogar einen Offset vorgeben. Diesen können Sie so benutzen, dass Sie die Koordinaten vorgeben und dann eine feste Verschiebung (Offset) angeben. Dies kann sinnvoll sein, wenn Sie einen Extrempunkt benennen wollen.

set label “Text“ at 1, 2 rotate by 20 textcolor rgbcolor “royalblue“

Die Farbnamen von Gnuplot erhalten Sie indem Sie Gnuplot starten und **show colornames** eintippen.

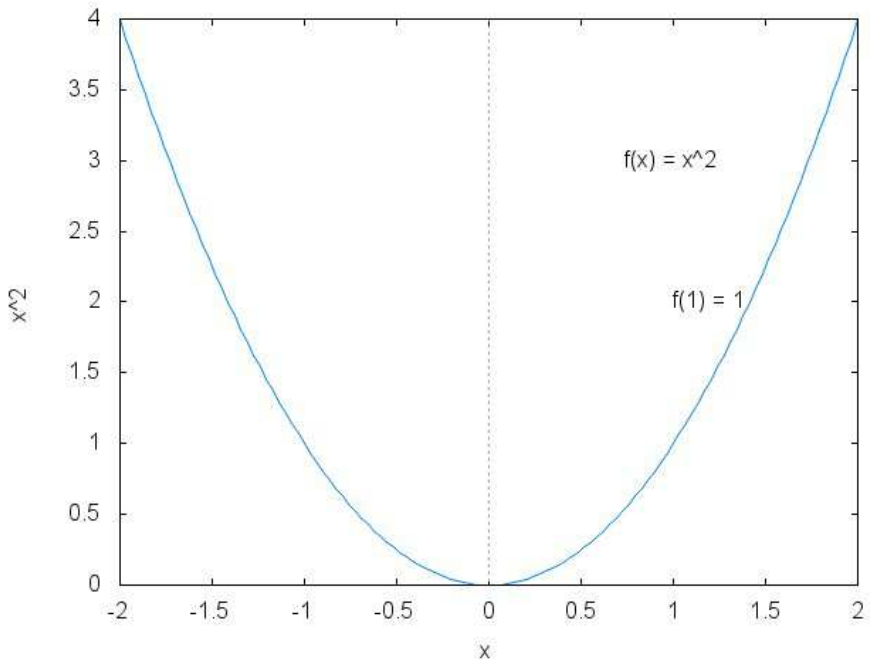
In **plot2d** (. . .) können Sie leider nur eine Zeichenkette übergeben, da die Optionen sehr beschränkt sind. Das heisst konkret, dass Sie alle Gnuplot-Befehle mit einem Semikolon abschliessen müssen und anschliessend zu einer Zeichenkette zusammenfügen müssen mit **concat** (*Zeichenkette, Zeichenkette*).

Übergeben Sie Ihre Zeichenkette dann mit **[gnuplot_preamble, gnuplotstr]** als Option zu **plot2d** (. . .).

Beachten Sie, dass Gnuplot selber kein jpg sondern nur jpeg kennt!

```
(%i1) f(x) := x^2;
                                                2
(%o1)                                    f(x) := x
(%i2) str1 : concat ("set label 'f(1) = ", string ( f(1) ) );
(%o2)                               set label 'f(1) = 1
(%i3) str1 : concat (str1, "' at 1, 2;");
(%o3)                           set label 'f(1) = 1' at 1, 2;
(%i4) str2 : "set label 'f(x) = x^2' at 1, 3 center;" ;
(%o4)                      set label 'f(x) = x^2' at 1, 3 center;
(%i5) gnuplotstr : concat (str1, str2);
(%o5) set label 'f(1) = 1' at 1, 2;set label 'f(x) = x^2' at 1, 3 center;
(%i6) plot2d (f(x),
        [x, -2, 2],
        [gnuplot_out_file, "graphiklabel2.jpg"],
        [gnuplot_term, jpeg],
        [gnuplot_preamble, gnuplotstr]
        );
```

Abbildung 13.5: Der Code ist hier: 346



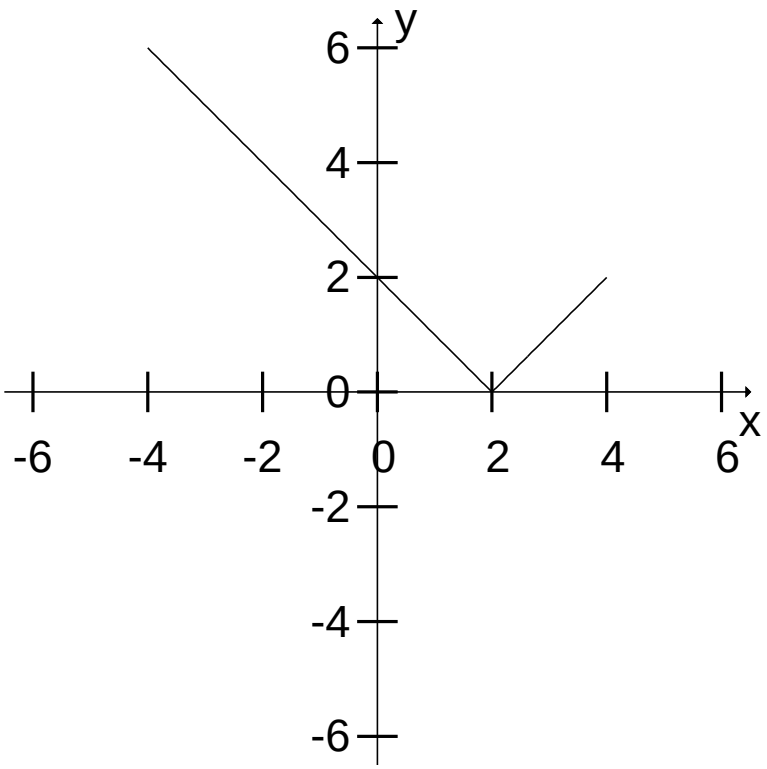
13.5 Eine Funktion stückweise zeichnen lassen

Sie haben eine Funktion stückweise definiert:

$$f(x) = \begin{cases} -x+2, \text{ für } x < 2 \\ x-2, \text{ für } x > 2 \end{cases}$$

```
(%i1) load (draw)$
(%i2) f(x) := -x+2$
(%i3) g(x) := (x-2)$
(%i4) draw2d (
    file_name = "stueckweise",
    terminal = jpg,
    color = black,
    user_preamble = ["set noborder"],
    dimensions = [500, 500],
    /* Koordinatensystem */
      head_length = 0.1,           /* Pfeil-Kopflaenge */
      vector ([-6.5, 0], [13, 0]),   /* x-Achse */
      vector ([0, -6.5], [0, 13]),   /* y-Achse */
      label (["x", 6.5, -0.5], ["y", 0.5, 6.5]),
      xtics_axis = true,
      ytics_axis = true,
      xtics = [-6, 2, 6],
      ytics = [-6, 2, 6],
    xrange = [-7, 7],
    yrange = [-7, 7],
    explicit (
      if x < 2 then
        f(x)
      else
        g(x),
      x, -4, 4
    )
  )$
```

Abbildung 13.6: Der Code ist hier: 347



Sie laden mit **load** (*draw*) das Paket draw zum Zeichnen.

Mit **file_name** geben Sie den Dateinamen des zu erstellenden Bildes an.

terminal gibt den Bildtyp an. In diesem Fall ein jpg-Bild.

Mit **color** = black wird alles nachkommende schwarz gemalt.

user_preamble übergibt diesen Befehl direkt an Gnuplot und sorgt in diesem Fall dafür, dass um das Bild herum kein Rahmen gezeichnet wird.

Dann erstellen wir das Koordinatensystem. Wenn Sie **xaxis** verwenden, wird eine gestrichelte Linie gezeichnet. Wir wollen aber ein durchgehend gezeichnetes Koordinatensystem. Dazu werden Pfeile mit **vector** gezeichnet. Damit die Pfeile nicht zu groß sind, müssen Sie die Pfeilgröße mit **head_length** einstellen. Die 0.1 sind relative Koordinaten. Bei einem Bild mit einem großen x-, y-Bereich ist muss head_length entsprechend größer eingestellt werden.

An **vector** werden zwei Argumente übergeben:

1. Startwert: [x, y]
2. Veränderung: [Δx, Δy]

Sie müssen also jeweils die ganze Länge des Vektors eintragen.

xtics reagiert unterschiedlich je nachdem, was das Argument ist:

1. xtics = none,
   Keine Tics werden auf der x-Achse gesetzt.
2. xtics = 2,
   Dies ist der Abstand zwischen den einzelnen Tics. Der Beginn und das Ende werden von Gnuplot automatisch gesetzt.

3. xtics = auto,
   So hat xtics kein Argument und wird automatisch gesetzt. Dies ist in den meisten Fällen gut und auf jeden Fall ein guter Startpunkt bei der Bilderstellung. Dies ist der Standardwert.

4. xtics = [-6, 2, 6],
   Hier wird eine Liste bestehend aus drei Elementen übergeben. Der erste Wert ist der Startwert, dann kommt der Wert, der zwischen den einzelnen Tics sein soll und zum Schluss der Wert für den letzten Tic.

5. xtics = -6, 3, 1, 5,
   Die Tics werden an der Stelle x = -6, x = 3, x = 1 und x = 5 gezeichnet.

6. xtics = [“Bananen“, 1], [“Gurken“, 2],
   führt zu zwei Tics, bei x = 1 und bei x = 2. Die Beschriftung ist aber dann „Bananen“ und „Gurken“.

Durch die **xrange** und **yrange** Angabe wird das Bild von jeweils von -7 bis 7 gezeichnet. Da die Funktion nur über einen kleineren Bereich gezeichnet wird, entsteht so ein Randeffekt. Dies zeigt die Möglichkeiten der verschiedenen Optionen.

explicit zeichnet jetzt die Funktion. Diese wird jetzt auf unterschiedlichen Abschnitten definiert durch die **if**-Anweisung:
Wenn x kleiner als 2 ist, dann
zeichne f(x)
sonst
g(x)
x ist die Variable, welche von -4 bis 4 gezeichnet werden soll.

13.6 Veranschaulichen der Transformation einer Matrix

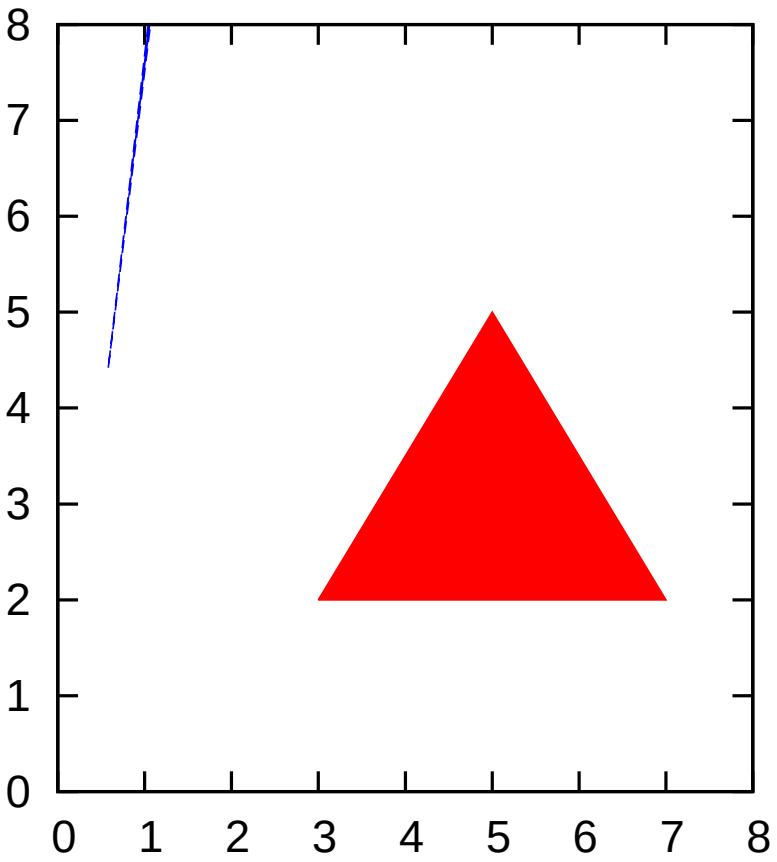
Sie haben eine Matrix und möchten die Transformation gerne veranschaulichen. In unserem Beispiel haben wir eine stochastische Matrix und können den Grenzwertprozess bei steigender Potenz von M verfolgen.

Sie können beim Ausprobieren die Potenz variieren. Hier ist die Potenz der Matrix auf das Quadrat also 2 festgelegt.

Vergleichen Sie dieses Beispiel auch mit: 12.6 auf S. 123.

```
(%i1) load(draw)$
(%i2) M : matrix ([0.2, 0.1], [0.8, 0.9])$
(%i3) X : matrix ([x], [y])$
(%i4) M.X;
                                [ 0.1 y + 0.2 x ]
(%o4)                           [               ]
                                [ 0.9 y + 0.8 x ]
(%i5) my_gnuplot_preamble : [
    "set yrange [0:8]",
    "set xrange [0:8]"
    ];
(%o5)                 [set yrange [0:8], set xrange [0:8]]
(%i6) draw2d(
        file_name = "graphikmatrix",
        terminal = jpg,
        dimensions  = [500, 500],
        user_preamble = my_gnuplot_preamble,
        color = "#e245f0",
        line_width = 8,
        border     = false,
        triangle([3,2],[7,2],[5,5]),
        fill_color = blue,
        transform  = [
                      (M^^2 . X)[1][1],
                      (M^^2 . X)[2][1],
                      x, y
                     ],
        triangle([3,2],[7,2],[5,5]) )$
```

Abbildung 13.7: Der Code ist hier: 348



Zuerst wird das Paket **load** (*draw*) geladen, damit Sie **draw** benugzen können.

Dann wird die Matrix definiert. Darüber hinaus wird eine zusätzliche Matrix X definiert mit den Variablen x und y.

M.X ergibt dann gerade eine 1-spaltige Matrix bei der die Matrixwerte als Vorfaktoren der Variablen erscheinen.

Im gnuplot_preamble wird festgelegt, dass das Bild im y-Bereich von 0 bis 8 und im x-Bereich ebenfalls von 0 bis 8 geht.

Dies könnten Sie auch mit draw machen indem Sie angeben:
xrange = [0, 8], yrange = [0, 8]

- file_name gibt die Datei an, in die das Bild geschrieben wird.
- terminal = jpg ist das Dateiformat.
- pic_width und pic_height gelten für gif, png, jpg und svg Dateien und gibt die Ausmaße des Bildes an.
- user_preamble ist die Liste der gnuplot-Befehle.
- color gibt die Farbe an, in der künftig gezeichnet wird.
- line_width gibt die Linienstärke an.
- border gibt an, ob ein Rahmen um die Objekte also hier das Dreieck gezeichnet werden soll.
- triangle zeichnet ein Dreieck.
- fill_color ist die Farbe in der zukünftig Objekte eingefärbt werden.

- transform verändert die Koordinaten.

transform (f(x,y), f(x,y), x, y) verändert im zweidimensionalen die Koordinatenachsen. (M^^4 . X)[1] gibt Ihnen die erste Liste der Matrix. [1][1] gibt Ihnen den Inhalt des Elements in der 1. Zeile und 1. Spalte.

Sie sehen, dass das Dreieck gerade auf den Eigenvektor $\left(\begin{smallmatrix}1\\8\end{smallmatrix}\right)$ abgebildet wird.

13.7 Veranschaulichen der Substitution bei Funktionen

Sie wollen die Substitution und ihre Auswirkungen auf den Graphen (und seine Ableitung) visualisieren.

Dazu benutzen Sie **transform**. Dies ist eine Option bei **draw2d**(...) und **draw3d**(...). Wir besprechen hier nur die Option von **draw2d**(...). **draw3d**(...) ist dann analog.

transform = [f(x, y), g(x, y), x, y] x, y gibt die Variablennamen für den x-Wert und den y-Wert für die transformierenden Funktionen (f(x) und g(x)) an. Sie könnten auch schreiben: **transform = [a+1, b, a, b]**. Dann steht a für die x-Koordinate und b für die y-Koordinate. In diesem Beispiel wird die x-Koordinate immer um eins vergrößsert.

Die erste Funktion transformiert die x-Koordinate und die zweite Funktion transformiert die y-Koordinate.

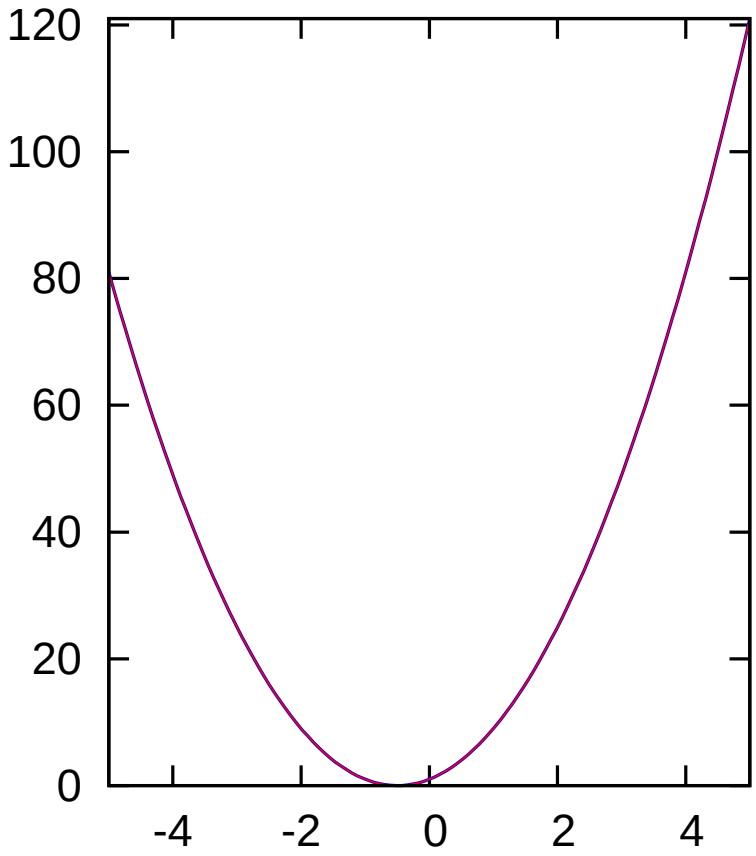
Sie haben folgende Funktion gegeben:

$$\begin{aligned} f(x) &= (2x+1)^2 \\ z &= 2x+1 \\ f(z) &= z^2 \end{aligned}$$

Mit einem Trick können Sie transform anschaulich verwenden. (Dies ist nicht ganz im Sinne des Erfinders ...): Weisen Sie die äussere Funktion der y-Koordinate in **transform** zu.

```
(%i1) load (draw)$
(%i2) draw2d (
    file_name = "graphiktransformfunktion",
    terminal = jpg,
    dimensions = [500, 500],
    line_width = 2,
    explicit ((2*x+1)^2, x,-5, 5),
    color = red,
    line_width = 1,
    transform = [x, y^2, x, y],
    explicit (2*x+1, x, -5, 5)
)$
```

Abbildung 13.8: Der Code ist hier: 349



Zur besseren Sicht ist der Graph f(x) in schwarz und dickerer Linienstärke und der Graph f(z) in rot und dünnerer Linienstärke gezeichnet.

Bemerkung:
Wenn Sie im Zeichen-Befehl **explicit (f(x), Variable, min, max)** die äussere Funktion schreiben wollen und in transform die innere Funktion, dann müssen Sie die x-Koordinate mit der Umkehrfunktion von z(x) transformieren:
transform = [(x-1)/2, y, x, y],
explicit (x^2, x,-5, 5)

13.8 Funktionenscharen als animiertes Gif zeichnen

Gegeben ist die Funktionenschar:

$$f_k(x) = x^4 - kx^2$$

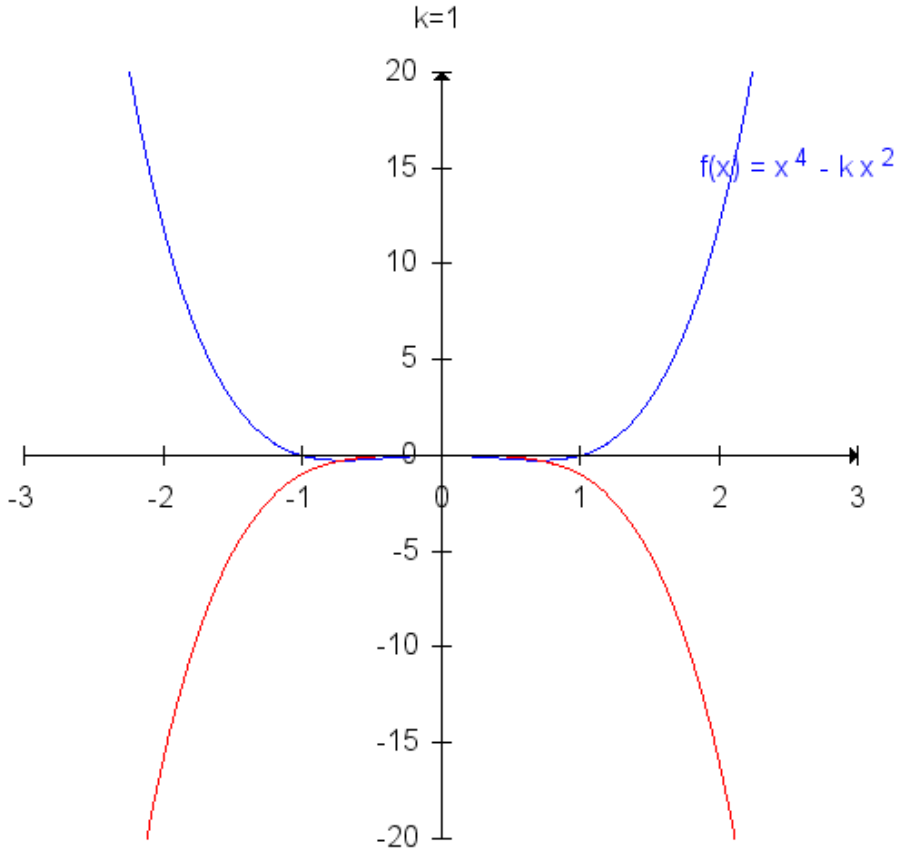
mit der Ortskurve:

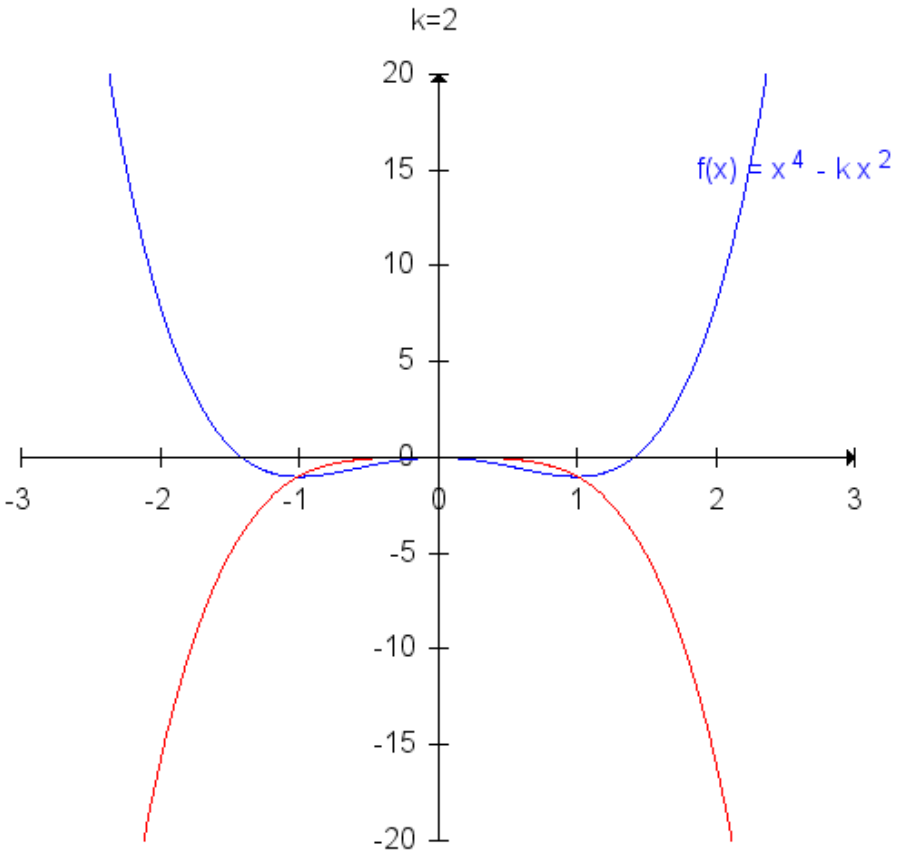
$$ye(x) = -x^4$$

Sie möchten die Ortskurve zeichnen und einige ausgewählte Graphen der Funktionenschar.

```
(%i1) load (draw)$
(%i2) f(x, k) := x^4 - k * x^2 $
(%i3) ye(x) := -x^4 $
(%i4) my_gnuplot_preamble : [
    "set noborder"
    ]$
(%i5) set_draw_defaults (
    user_preamble = my_gnuplot_preamble,
    yrange = [-20, 20],
    xtics_axis = true,
    ytics_axis = true,
    xaxis       = true,
    yaxis       = true
    )$
(%i6) graph2 : [
            label(["f(x) = x^4 - k x^2", 2.6, 15]),
            color = red, explicit ( ye(x)  , x, -3, 3),
            color = black,
            head_length = 0.1,
            vector ( [-3, 0], [6, 0] ),
            vector ( [0, -20], [0, 40] )
         ]$
(%i7) bild1 : gr2d (title="k=1",
    [explicit ( f(x, 1), x, -3, 3)], graph2
    )$
(%i8) bild2 : gr2d (title="k=2", [explicit (f(x, 2), x, -3, 3) ], graph2)$
(%i9) bild3 : gr2d (title="k=3", [explicit (f(x, 3), x, -3, 3) ], graph2)$
(%i10) bild4 : gr2d (title="k=4", [explicit (f(x, 4), x, -3, 3) ], graph2)$
(%i11) bild5 : gr2d (title="k=5", [explicit (f(x, 6), x, -3, 3) ], graph2)$
(%i12) bild6 : gr2d (title="k=6", [explicit (f(x, 6), x, -3, 3) ], graph2)$
(%i13) bild7 : gr2d (title="k=7", [explicit (f(x, 7), x, -3, 3) ], graph2)$
(%i14) bild8 : gr2d (title="k=8", [explicit (f(x, 8), x, -3, 3) ], graph2)$
(%i15) draw (
    delay = 80,
    dimensions = [300, 300],
    file_name = "funktionenschar1",
    terminal = 'animated_gif,
    bild1, bild2, bild3, bild4, bild5, bild6, bild7, bild8
    )$
End of animation sequence
```

Abbildung 13.9: Der Code ist hier: 350





Laden Sie das Paket **load** (*draw*) damit Sie den Befehl draw zur Verfügung haben. Definieren Sie ansch-

liessend die beiden Funktionen. Damit Sie k leichter eingeben können, empfiehlt es sich, f als eine Funktion mit den Variablen x und k zu definieren.

Damit Ihre Zeichnung keinen Rand aufweist, wird der Gnuplot-Befehl set noborder übergeben.
Als nächstes werden die zukünftigen Standardwerte für das Zeichnen definiert.

- user_preamble = my_gnuplot_preamble,
  Die Gnuplot-Befehle sind als Zeichenketten in der Liste my_gnuplot_preamble abgespeichert und werden vor dem zeichnen in Gnuplot eingefügt.

- yrange = [-20, 20],
  Diese Option legt die Spannweite für die y-Werte fest.

- xtics_axis = true,
  Die xtics werden an die x-Achse geschrieben und nicht an den Rand.

- ytics_axis = true,
  Die ytics werden an die y-Achse geschrieben und nicht an den Rand.

- xaxis = true,
  Die x-Achse wird eingezeichnet. Aber nur als gepunktete Linie. Wenn Sie ein Koordinatenkreuz haben wollen, müssen Sie noch Pfeile (**vector** (*[x, y], [dx, dy]*)) zeichnen lassen.

- yaxis = true,
  Die y-Achse wird als gepunktete Linie eingezeichnet.

Nun werden alle gleichbleibenden Objekte der Bilder in graph2 definiert:

- label([“f(x) = x^4 - k x^2“, 2.6, 15]),
  Schreibt eine Markierung (label) an die Koordinaten 2,6 und 15.

- color = red
  Die zukünftige zu benutzende Farbe ist rot.

- explicit (ye(x) , x, -3, 3),
  Die Ortskurve wird angegeben mit Hilfe von **explicit** (*f, Var, min, max*).

- color = black
  Die zukünftige zu benutzende Farbe ist schwarz.

- head_length = 0.1,
  Setzt die Pfeillänge des Vektors in Koordinaten.

- vector ([-3, 0], [6, 0]),
  vector (*[x, y], [dx, dy]*) x, y ist der Beginn des Vektors dx, dy.

Nun werden die einzelnen Bilder (oder auch Szenen) definiert.

```
bild1 : gr2d (title="k=1",
    [explicit ( f(x, 1), x, -3, 3)], graph2
    )$ %$
```

gr2d ist das Graphikobjekt. Hier werden alle sich änderenden Optionen übergeben und graph2 als Liste alle gleichbleibenden Optionen.

```
draw (
    delay = 80,
    file_name = "funktionenschar1",
    terminal = 'animated_gif,
    bild1, bild2
    );
```

- draw
  draw (*gr2d, . . . , gr3d, . . . , Optionen*) zeichnet Ihre Bilder.
- delay = 80
  Die Zeit in Millisekunden zwischen jedem einzelnen Bild.
- file_name = “funktionenschar1“,
  Ausgabename der Datei.
- terminal = ’animated_gif,
  Typ der Datei.
- pic_width = 300,
  und: pic_height = 300, (denken Sie an das Komma) geben die Breite und Höhe des Bildes an.
- bild1, bild2 Hier übergeben Sie die einzelnen Bilder zum Zeichnen.

13.9 Eine Fläche einfärben, welche von zwei Funktionen begrenzt wird.

Sie haben zwei Funktionen, die eine Fläche einschliessen. Färben Sie diese Fläche ein.

13.9.1 Fläche unter einer Funktion

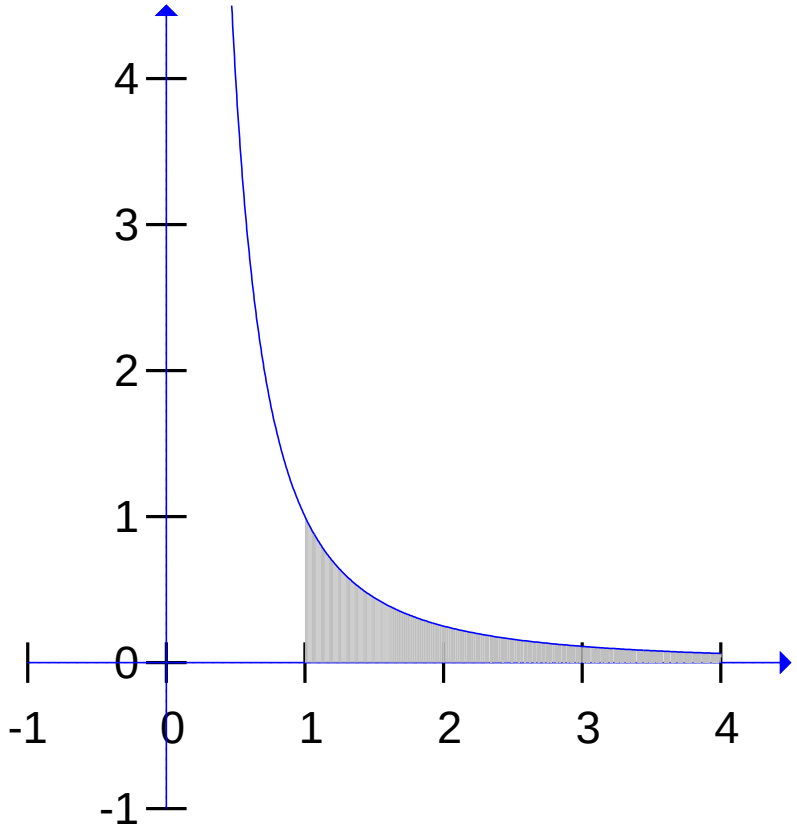
Sie möchten die Fläche zwischen der Funktion f und der x-Achse:

$$f(x) = \frac{1}{x^2}$$

im Intervall [1;4] einfärben. Trotzdem soll die Funktion aber in einem größeren Intervall [0,2; 4] gezeichnet werden.

```
(%i1) load (draw)$
(%i2) f(x) := 1/x^2$
(%i3) g(x) := 0$
(%i4) draw2d (
    /* Allgemeiner Bildaufbau */
    user_preamble = ["set noborder"],
    border = false,
    dimensions = [500, 500],
    terminal = jpg,
    file_name = "gefuellteFunktion_einfach",
    /* Koordinatensystem */
    xaxis = true,
    yaxis = true,
    xtics_axis = true,
    ytics_axis = true,
    head_length = 0.1,
    vector ([-1, 0], [5.5, 0]),
    vector ([0, -1], [0, 5.5]),
    xrange = [-1, 4.5],
    yrange = [-1, 4.5],
    /* Funktionen */
    fill_color = grey,
    filled_func = g(x),
    explicit (f(x), x, 1, 4),
    filled_func = false,
    explicit (f(x), x, 0.2, 4)
)$
```

Abbildung 13.10: Der Code ist hier: 351



filled_func füllt den Bereich zwischen der Funktion und der Unterkante des Fensters, also in diesem Fall bis $y = -1$. Damit die Fläche nur bis zur x-Achse gefüllt wird, müssen Sie entweder **yrange** bei 0 beginnen lassen oder die x-Achse als untere begrenznde Funktion ($g(x) = 0$) angeben.

filled_func = g(x) bildet eine Grenze. Die anschliessend mit **explicit** gezeichnete Funktion bildet die andere Grenze.

Anschliessend wird das Füllen mit **filled_func = false** ausgeschaltet und die Funktion wird über einen größeren Bereich gezeichnet.

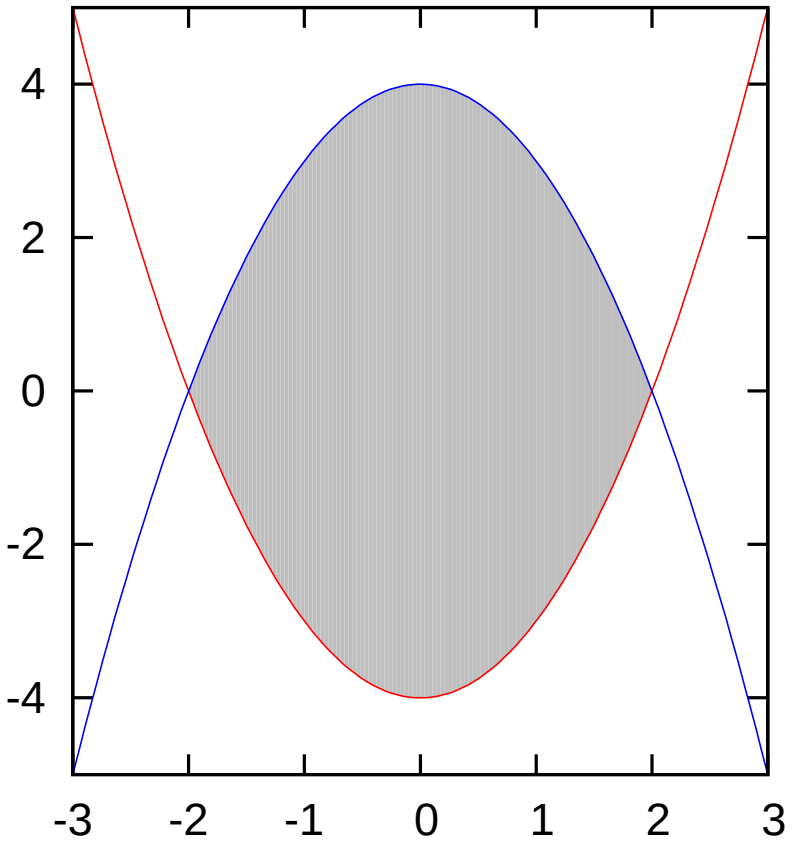
13.9.2 Lösung mit Beschriftung am Rand

Normalerweise werden die Koordinaten an den Rand geschrieben. Hier ist es so, dass gnuplot die Koordinatenachsen hinter der gefüllten Fläche malt. So sind dann die Koordinatenachsen und deren Beschriftung unterbrochen, wenn Sie sie von gnuplot in der Mitte des Bildes zeichnen lassen. Sie können also entweder die Achsenbeschriftung an den Rand legen oder versuchen die Achsen und die Beschriftung von Hand mit vielen Labeln konstruieren.

Es gibt jedoch einen Trick, wie Sie die Achsenbeschriftung automatisch von Gnuplot erstellen lassen. Dieser Trick ist leider unter Windows nicht möglich. Sie benutzen die multiplot-Option. Dann können Sie sich das Bild auf den Bildschirm zeichnen lassen und dann in die Zwischenablage kopieren und dort z. B. in Gimp einfügen.

```
(%i1) load (draw)$
(%i2) f(x) := x^2 - 4$      g(x) := -x^2 + 4$
(%i4) [x1, x2] : map( rhs, solve(f(x) = g(x)));
(%o4)                                [- 2, 2]
(%i5) bild1 : [
      fill_color = grey,
      filled_func = g(x),
      explicit (f(x), x, x1, x2),
      filled_func = false
    ]$
(%i6) bild2 : [
    color = red,
    explicit(f(x), x, -3, 3),
    color = blue,
    explicit(g(x), x, -3, 3)
    ]$
(%i7) set_draw_defaults (
    dimensions = [500, 500],
    xtics_axis = false,
    ytics_axis = false
    )$
(%i8) draw ( gr2d (
      file_name = "gefuellteFunktion_normal", terminal = jpg,
      bild1, bild2
      )
     )$
```

Abbildung 13.11: Der Code ist hier: 352



Sie laden mit **load** (*draw*) das Zeichenpaket.

Zuerst berechnen Sie die Schnittpunkte der beiden Funktionen mit **solve** (*Gleichung, Variable*). Bei solve erhalten Sie eine Liste mit den Elementen: x=-2 und als zweites Element x=2. **rhs** (*Ausdruck*) gibt Ihnen die rechte Seite des Ausdrucks zurück. **map** (*Funktion, Liste*) führt rhs auf jedes Element der Liste aus und gibt eine Liste (mit zwei Elementen) zurück, welche auf x1 und x2 zugewiesen wird.

Sie definieren nun zwei Bilder.

1. Im ersten Bild werden die beiden Funktionen gezeichnet zwischen den Schnittpunkten. filled_func = g(x) zeichnet die Fläche zwischen g(x) und einer weiteren Funktion gefüllt. Im ersten Bild werden keine xtics und ytics gezeichnet.

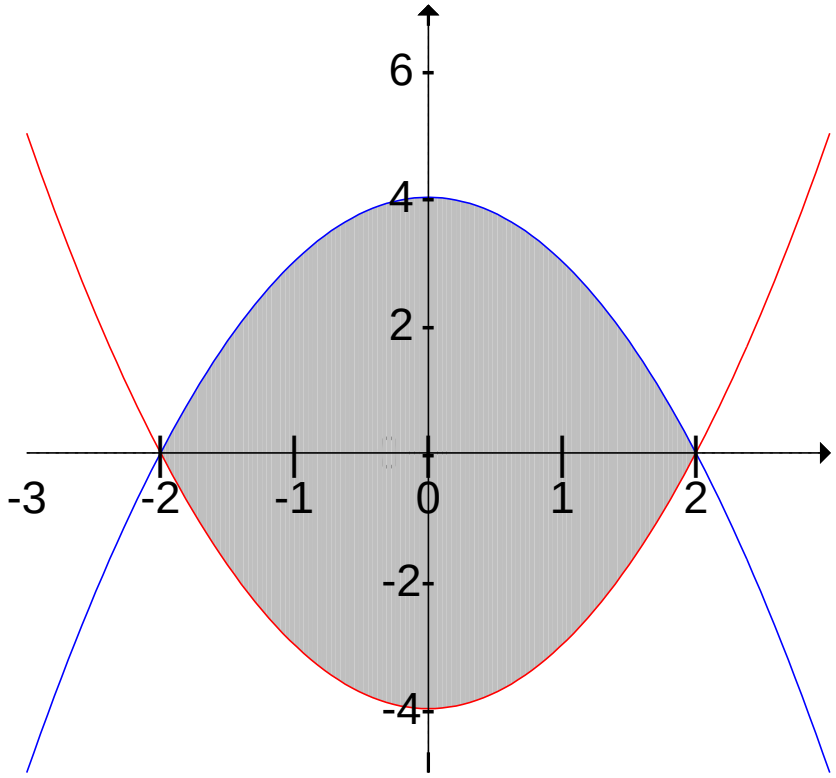
2. Im zweiten Bild werden die Funktionen farbig gezeichnet, über einen weiteren Bereich. Es werden ebenfalls keine xtics oder ytics gezeichnet.

Die Beschriftung erfolgt aussen am Rand.

13.9.3 Lösung mit selbstgebauten Koordinatenbeschriftung

```
(%i1) load (draw)$
(%i2) f(x) := x^2 - 4$      g(x) := -x^2 + 4$
(%i4) [x1, x2] : map( rhs, solve(f(x) = g(x)));
(%o4)                              [- 2, 2]
(%i5) bild1 : [
      fill_color = grey,
      filled_func = g(x),
      explicit (f(x), x, x1, x2),
      filled_func = false
    ]$
(%i6) bild2 : [
    color = red, explicit(f(x), x, -3, 3),
    color = blue, explicit(g(x), x, -3, 3)
    ]$
(%i7) koord : [
    head_length = 0.1,           /* Pfeil-Kopflaenge */
    color = black,
    vector ([-3, 0], [6, 0]),    /* x-Achse */
    vector ([0, -5], [0, 12]),   /* y-Achse */
    xtics = {["", 0]}, ytics = {0},    /* Bei (0|0) sieht man nichts */
    label ( /* xtics */
    ["-3", -3, -0.7], ["-2", -2, -0.7], ["-1", -1, -0.7],
    ["0", 0, -0.7],   ["1", 1, -0.7],   ["2", 2, -0.7],
        /* ytics */
    ["-4", -0.2, -4], ["-2", -0.2, -2], ["2", -0.2, 2],
    ["4", -0.2, 4],   ["6", -0.2, 6],
        /* Markierungen */
    ["|",-2,0], ["|",-1,0], ["|",0,0], ["|",1,0], ["|",2,0], ["-",0,-4],
    ["-",0,-2], ["-",0,0], ["-",0,2], ["-",0,4],  ["-",0,6]
        ) ]$  /* Nur noetig, wenn die Flaeche die x-Achse ueberstreicht */
(%i8) set_draw_defaults (
(%i8)     border = false,
(%i8)     dimensions = [500, 500],
(%i8)     xaxis = true, xtics_axis  = false, yaxis = true, ytics_axis  = true
(%i8)     )$
(%i9) draw2d (
      file_name = "gefuellteFunktion", terminal = jpg,
      bild1, bild2, koord,
      user_preamble = ["set noborder"]
      )$
```

Abbildung 13.12: Der Code ist hier: 353



Dieses 2. Beispiel ist etwas umfangreich und kompliziert, weil es den ungünstigsten Fall darstellt. Da hier die Option **filled_func** benutzt wird, wird die Fläche zwischen f(x) und g(x) grau gefüllt. Leider wird dabei auch unwiederuflich die x-Achse und ihre Markierungen übermalt. In diesem Beispiel wird gezeigt, wie man sich die Markierung wieder herstellen kann. Wenn die zu füllende Fläche über der x-Achse und neben der y-Achse liegt, muss man diesen Aufwand nicht treiben. Ebenso muss man die Markierungen nicht mehr setzen, wenn man die Markierungen am Rand beläßt, wie es auch die Standardeinstellung ist.

Aus Platzgründen werden die Befehle hier mit dem Dollarzeichen beendet. Dies ist für den normalen Gebrauch sehr ungünstig, weil Sie dann Fehler unter Umständen erst spät bemerken.

Zuerst wird das Paket **load** (*draw*) geladen.

Die beiden die Fläche begrenzenden Funktionen werden definiert.

rhs (*Gleichung*) gibt die rechte Seite einer Gleichung zurück. Hier werden in x1 und x2 die x-Werte der Schnittpunkte der beiden Funktionen gespeichert.

Nun werden die einzelnen Bilder einzeln definiert. Das dient hier der Übersicht, ist aber nicht zwingend erforderlich. Da alle Bilder einem Szenenkonstruktor (gr2d) zugeordnet werden, sind hier nur die Optionen gespeichert.

bild1: Die Füllfarbe wird mit grau angegeben. filled_func wird eine Funktion zugewiesen. Standardmäßig ist die x-Achse die begrenzende Linie. Jetzt wird g(x) eine Begrenzung. Die andere Begrenzung ist die nächste zu zeichnende Funktion.

explicit (*Funktion, Variable, Startwert, Endwert*) zeichnet die Funktion.

bild2: Hier werden die Funktionen noch einmal aber in unterschiedlichen Farben gezeichnet.

koord: Hier wird das Koordinatensystem erstellt. **vector** (*[x, y], [dx, dy]*) zeichnet einen Vektor. Die Pfeillänge muss vorher mit head_length eingestellt werden.

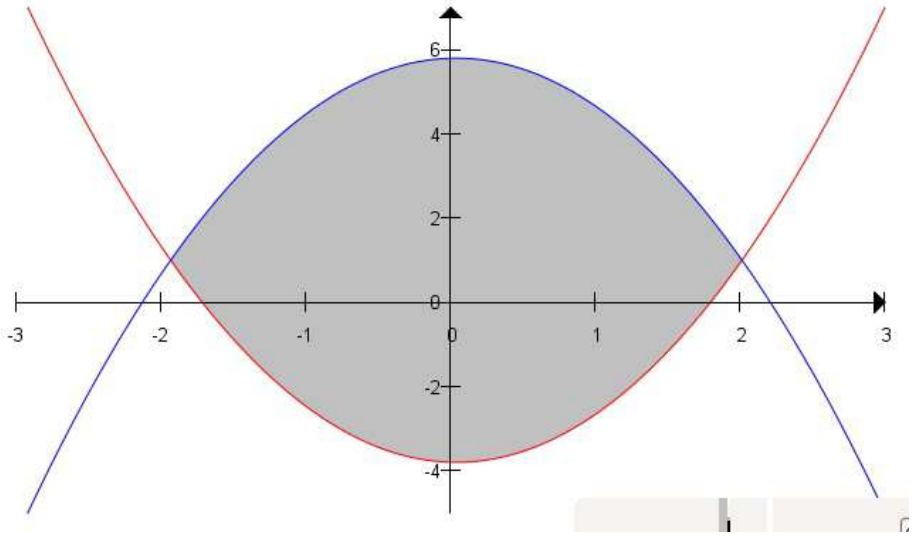
Es wird ein xtics und ein ytics gezeichnet. In draw_defaults sollen die xtics auf der x-Achse gezeichnet werden. Damit jetzt aber keine Mischung der selbsterstellten Markierungen und der automatischen Markierungen erfolgt, wird jetzt ein xtic gemalt, welches aufgrund der gefüllten Fläche sowieso nicht zu sehen ist.

draw (*gr2d, . . . Optionen*) zeichnet dann die Bilder in einer Szene. Die Optionen **pic_width** und **pic_height** geben die Breite und Höhe des Bildes an.

13.9.4 Lösung mit überlagerten Bildern

```
(%i1) load (draw)$
(%i2) f(x) := x^2 - 4$      g(x) := -x^2 + 4$
(%i4) [x1, x2] : map ( rhs, solve (f(x) = g(x) ) );
(%o4)                                [- 2, 2]
(%i5) bild1 : [
      fill_color = grey,
      filled_func = g(x),
      explicit ( f(x), x, x1, x2 ),
      filled_func = false,
      xtics = false, ytics = false
    ]$
(%i6) bild2 : [
    color = red,
    explicit ( f(x), x, -3, 3 ),
    color = blue,
    explicit ( g(x), x, -3, 3 ),
    xtics = false, ytics = false
    ]$
(%i7) koord : [
    head_length = 0.1,           /* Pfeil-Kopflaenge */
    color = black,
    vector ([-3, 0], [6, 0]),    /* x-Achse */
    vector ([0, -5], [0, 12]),   /* y-Achse */
    xaxis = true, xtics_axis  = true,
    yaxis = true, ytics_axis  = true
    ]$
(%i8) set_draw_defaults (
    border = false,
    dimensions = [300, 300]
    )$
(%i9) multiplot_mode(wxt)$
(%i10) draw2d (
       bild1, bild2,
       user_preamble = ["set noborder"]
       )$
(%i11) draw2d (
       koord,
       user_preamble = ["set noborder"]
       )$
(%i12) multiplot_mode(none)$
```

Abbildung 13.13: Der Code ist hier: 354



Der Code öffnet Ihnen ein Fenster, welches Sie in ein Graphikprogramm wie Gimp überführen und dort dann abspeichern können. Bei der Zeichnung wurden mehrere Bilder in das Fenster gezeichnet.

Sie laden mit **load** (*draw*) das Zeichenpaket.

Zuerst berechnen Sie die Schnittpunkte der beiden Funktionen mit **solve** (*Gleichung, Variable*). Bei solve erhalten Sie eine Liste mit den Elementen: x=-2 und als zweites Element x=2. **rhs** (*Ausdruck*) gibt Ihnen die rechte Seite des Ausdrucks zurück. **map** (*Funktion, Liste*) führt rhs auf jedes Element der Liste aus und gibt eine Liste (mit zwei Elementen) zurück, welche auf x1 und x2 zugewiesen wird.

Sie definieren nun drei Bilder. Das Problem ist, dass die gefüllte Fläche die x-Achse überdeckt. Dadurch wird die Beschriftung der x-Achse in dem Bereich ausgesetzt.

1. Im ersten Bild werden die beiden Funktionen gezeichnet zwischen den Schnittpunkten. filled_func = g(x) zeichnet die Fläche zwischen g(x) und einer weiteren Funktion gefüllt. Im ersten Bild werden keine xtics und ytics gezeichnet.

2. Im zweiten Bild werden die Funktionen farbig gezeichnet, über einen weiteren Bereich. Es werden ebenfalls keine xtics oder ytics gezeichnet.

3. Im dritten Bild wird das Koordinatensystem gezeichnet. Dort werden die Achsen als Pfeile gezeichnet. Die x-Achse wird durch den Nullpunkt und nicht am Rand gezeichnet durch xaxis=true. Die xtics werden an der Achse gezeichnet durch xtics_axis=true.

multiplot_mode funktioniert nicht unter Windows und nur mit den Optionen „none“, „screen“ und „wxt“. Wählen Sie wxt, dann können Sie das Bild herauskopieren und in Gimp einfügen.

13.10 Visualisieren der Untersumme einer stetigen monotonen steigenden Funktion

Sie wollen die Untersumme einer stetigen monoton steigenden Funktion in einer Graphik visualisieren.

Hier werden Ihnen zwei Lösungen angeboten: Als animiertes gif und mit Hilfe des Schiebereglers.

Alle hier vorgestellten Lösungen bilden nur die Untersumme, wenn Sie eine stetige, monoton steigende Funktion haben. Wenn Sie auch bei anderen stetigen Funktionen die Untersumme bilden wollen, dann müssen Sie die Funktion f durch eine Funktion ersetzen, welche Ihnen den minimalen Wert in einem Intervall bildet (siehe dazu Kap. 6.7, S. 6.7).

13.10.1 Untersumme als animiertes gif

Sie wollen die Untersumme eine monoton steigenden Funktion darstellen:

$$f(x) = x^2 + 1$$

(Dies ist also die einfache Version der Untersumme. Vergleichen sie auch Kap. 13.11 auf S. 157)

```
(%i1) load (draw)$
(%i2) f(x) := x^2 + 1$
(%i3) start : 1$
(%i4) end : 4$
(%i5) n : [1, 2, 4, 8, 16, 32, 64, 128]$
(%i6)
s(n) :=
  block ([breite, l, i],
    breite : (end - start) / n,
    l : bars ([start + breite/2, f(start), breite]),
    for i : 1 while i < n do
      block ([x],
        xl : start + i * breite,
        l : append (l, bars ([xl + breite/2, f(xl), breite]))
      ),
    return (l)
  )$
(%i7)
for i : 1 while i < length(n) do
  block ([],
    bild[i] : gr2d (
      title = concat ("n = ", string(n[i])),
      xrange = [0, 4],
      yrange = [0, 20],
      explicit (f(x), x, 0, 4),
      s(n[i])
    )
  )$
(%i8)
bilder : [bild[1]]$
(%i9) for i : 2 while i < length(n) do
  bilder : append (bilder, [bild[i]])$
(%i10)
draw (
      delay = 80,
      dimensions = [500, 500],
      file_name = "graphikUntersummeMonoton",
      terminal = 'animated_gif,
      bilder
    )$
End of animation sequence
```

Abbildung 13.14: Der Code ist hier: 355

Zuerst wird das Paket mit **load** (*draw*) geladen. Dies brauchen Sie zum zeichnen.
Die Funktion und der Start- und Endwert der Balken werden festgelegt.
Dann werden in der Liste n die verschiedenen Anzahlen der Balken festgelegt.

Nun erfolgt die Funktion **s(n)**, welche die Balken (bars) erstellt. n ist die Anzahl der zu zeichnenden Balken. Zuerst wird die Breite der einzelnen Balken in breite abgespeichert. l wird die Liste der Balken. Der erste Balken wird der Liste zugeordnet, so dass Maxima auch weiss, dass l eine Liste ist. Dann wird eine for-Schleife für die einzelnen Balken benötigt. xl ist der linke x-Wert des Balkens. **bars** (*xpos der Mitte*, *Höhe*, *Breite*) erstellt den Balken. Da der Balken in der Mitte zentriert dargestellt wird, muss die Position der Mitte angegeben werden.

In den einzelnen Bildern wird der Graph der Funktion f gezeichnet und dazu die jeweiligen Balken.

draw (*gr2d*, *Optionen*, . . .) zeichnet dann die Bilder.

Die Optionen **pic_width** und **pic_height** geben die Breite und Höhe des Bildes an.

Der Unterschied zu der Lösung mit dem Schieberegler ist, dass Sie die einzelnen Bilder hier selbst erstellen müssen.

13.10.2 Mit Hilfe des Schiebereglers

Diese Lösung funktioniert nur mit Hilfe von Wxmaxima und nicht mit Maxima direkt, denn der Schieberegler ist nur in Wxmaxima vorhanden.

Die Untersumme der stetigen, monoton steigenden Funktion f soll erstellt werden:

$$f(x) = x^2 + 1$$

```
f(x) := x^2 + 1$
start : 1$
end : 4$
s(n) :=
  block ([breite, l, i],
    breite : (end - start) / n,
    l : bars ([start + breite/2, f(start), breite]),
    for i : 1 while i < n do
      block ([x],
        x : start + i * breite,
        l : append (l, bars ([x + breite/2, f(x), breite]))
      ),
    return (l)
  )$
with_slider_draw (n , [1, 2, 4, 8, 16, 32, 64],
      xrange = [0, 4],
      yrange = [0, 20],
      explicit (f(x), x, 0, 4),
      s(n)
    )$
```

Abbildung 13.15: Der Code ist hier: 356

Zuerst werden die Funktion und die Startwerte und Endwerte für die Untersumme festgelegt.

s(n) ist eine Funktion, welche n Balken zeichner im Intervall start bis end. Dazu wird erst die Breite festgelegt und in „breite“ abgespeichert.

l wird die Liste der einzelnen Säulen. Dazu muss einmal diese Liste angelegt werden, damit später die anderen Elemente mit append angehängt werden können.

append (*Liste, Liste, . . .*) fügt Liste zusammen.

Die eigentlich interessante Funktion ist **with_slider_draw** (*BildlaufVariable, Liste, Graphikoptionen*) n durchläuft nacheinander die Werte aus der nachfolgenden Liste. Genau so viele Bilder werden erstellt, die Sie mit dem Schieberegler (oder der Maus) dann einstellen können nachdem Sie das Bild einmal angeklickt haben.

13.11 Visualisieren der Untersumme einer stetigen Funktion

In diesem Abschnitt soll die Untersumme einer stetigen Funktion visualisiert werden. Diese muss nicht streng monoton steigend sein. Deshalb müssen Sie jeweils das Minimum aus einem Abschnitt bestimmen und dies als Höhe für Ihr Rechteck wählen. Im Vergleich: Bei einer monoton steigenden Funktion können Sie immer die links (bei einer steigenden Funktion) oder die rechte Ecke (bei einer fallenden Funktion) nehmen.

```
(%i1) load (draw)$
(%i2) f(x) := x * (x-2) * (x-4) + 10$
(%i3) start : 1$ end : 4$
(%i5) n : [1, 2, 4, 8, 16, 32, 64, 128]$
(%i6) minf(f, a, b) := block ([],  /* Minimum von f im Intervall [a,b] */
    df : diff (f, x),
    x_nst_lsg_ableitung : map (rhs, solve (df = 0, x)),
    xpos : append ([a], [b], x_nst_lsg_ableitung),
    xpos_Intervall_ind : sublist_indices (xpos,
                lambda ([x], x >= a and x <= b) ),
    xwerte_Intervall : makelist (xpos[x], x, xpos_Intervall_ind),
    g(x) := ev (f, 'x = x),
    ywerte : create_list (g(x), x, xwerte_Intervall),
    return ( lmin (ywerte) )
  )$
(%i7) s(n) := block ([breite, l, i],        /* Histogramme */
    breite : (end - start) / n,
    h : minf (f(x), start, start + breite),
    l : bars ([start + breite/2, h, breite]),
    for i : 1 while i < n do
      block ([x],
        xl : start + i * breite,
        h : minf (f(x), xl, start + (i+1) * breite),
        l : append (l, bars ([xl + breite/2, h, breite]))
      ),
    return (l)
  )$
(%i8) for i : 1 while i < length(n) do     /* Erstellt Liste von Bildern */
  block ([], bild[i] : gr2d (
      title = concat ("n = ", string(n[i])),
      xrange = [0, 4], yrange = [0, 20],
      explicit (f(x), x, 0, 4),
      s(n[i])
    ) )$
(%i9) bilder : [bild[1]]$
(%i10) for i : 2 while i < length(n) do
  bilder : append (bilder, [bild[i]])$
(%i11) draw ( delay = 80, dimensions = [500, 500],
      file_name = "graphikUntersumme", terminal = 'animated_gif,
      bilder
    )$
End of animation sequence
```

Abbildung 13.16: Der Code ist hier: 357

Die Anweisungen unterscheiden sich von Kap. 13.10.1, S. 154 durch die Wahl der Höhe der Balken.

- Die Funktion minf:

Die Balkenhöhe wird durch die Funktion minf erstellt. Vergleichen Sie diese Funktion auch mit Kap. funktionenMaxFunktionswertIntervall, S. 38. Diese Funktion liefert den kleinsten Wert einer Funktion f im Intervall a, b.

Die Idee dieser Funktion ist, dass der kleinste Wert entweder am Rand bzw. auf den Extrema liegt. Nun wollen wir aber nicht die Extrema bestimmen sondern nur die x-Werte, bei denen der Graph eine waagerechte Tangente hat. Alle die entsprechenden y-Werte werden dann verglichen und der Kleinste dieser Werte ist dann das Minimum im entsprechenden Intervall.

diff (*Ausdruck, Variable*) differenziert eine Funktion. Der Rückgabewert ist keine Funktion! Sie können also nicht df(x) schreiben sondern nur z. B.: df, x = 2.

solve (*Gleichung, Variable*) sucht hier die Nullstellen der 1. Ableitung. solve erstellt eine Liste mit den Elementen x = 0, x = 2 usw.

map (*f, Liste*) führt die erste Funktion auf die Liste aus. Hier wird also rhs auf die Liste mit den Gleichungen (von solve) ausgeführt.

rhs (*Ausdruck*) gibt die „rechte“ Seite eines Ausdrucks, Gleichung etc. Die Trennung der linken und rechten Seite kann durch verschiedene Symbole erfolgen wie $<$ oder $\leq$, equal etc. Damit haben Sie hier die einzelnen Werte der Lösungen extrahiert.

append (*Liste, Liste, . . .*) fügt die Listen zu einer einzigen Liste zusammen. Sie haben jetzt also in xpos eine Liste mit den Rändern und den x-Werten bei denen die Steigung null ist.

In xpos_Intervall_ind werden die Positionen der Liste xpos gesammelt, bei denen die x-Werte zwischen a und b liegen. Dazu wird die anonyme Funktion **lambda** (*[lokale Variable], Anweisungen*) benötigt. Jedes Listenelement von xpos wird an lambda (als x) übergeben. Wenn lambda „wahr“ zurückliefert, dann ist x zwischen a und b einschliesslich und diese Position wird in der Liste xpos_Intervall_ind festgehalten.

makelist (*Ausdruck, Variable, Liste*) erstellt eine neue Liste, welche genau so viele Elemente hat wie die Liste in makelist. Die Variable in dem Ausdruck (hier: x) nimmt nacheinander die Werte der Elemente der Liste in makelist an.

Da das übergebene f keine Funktion ist, wird aus dem Ausdruck durch diese Konstruktion mit Hilfe von **ev** (*Ausdruck*) eine Funktion erstellt. ’x bedeutet, dass das x nicht ausgewertet wird, sondern als Zeichen erhalten bleibt:

$$g(2) := ev(f, x = 2)$$

ev wertet dann den Ausdruck aus, d, h. in dem Ausdruck f wird jedes x durch 2 ersetzt.

create_list (*Funktion, Variable, Liste*) erstellt eine Liste. Jedes Element der Liste ersetzt die Variable in der Funktion. Hier wird jedes x in der Funktion g nacheinander durch die Elemente der Liste ersetzt.

lmin (*Liste*) gibt den kleinsten Wert einer Liste zurück.

- In der Funktion s(n) werden n gleichbreite Balken im Intevall von a bis b gezeichnet. l wird einmal eine Liste zugewiesen, so dass hinterher mit **append** (*Liste, Liste, . . .*) gearbeitet werden kann.

  h ist die Höhe der Balken, xl die Position der linken Ecke des Balken.

  bars (*[xpos, Höhe, Breite]*) zeichnet den Balken mit der Mitte bei xpos. Daher muss xpos um die Hälfte der Breite verschoben werden.

- Im nächsten Block werden die einzelnen Bilder erstellt.

  gr2d (*Graphikobjekt, Optionen, . . .*) erstellt jeweils ein Bild. Da wird animierte gifs haben wollen, müssen wir jede Szene in gr2d abspeichern.

  concat (*Zeichenkette1, Zeichenkette2*) fügt Zeichenketten zusammen.

- Dann werden die Bilder alle zusammengefügt.
- **draw** (*gr2d*) zeichnet die einzelnen Bilder.

13.12 Zeichnen eines Rotationskörpers

Sie möchten eine Flasche als Rotationskörper zeichnen lassen.

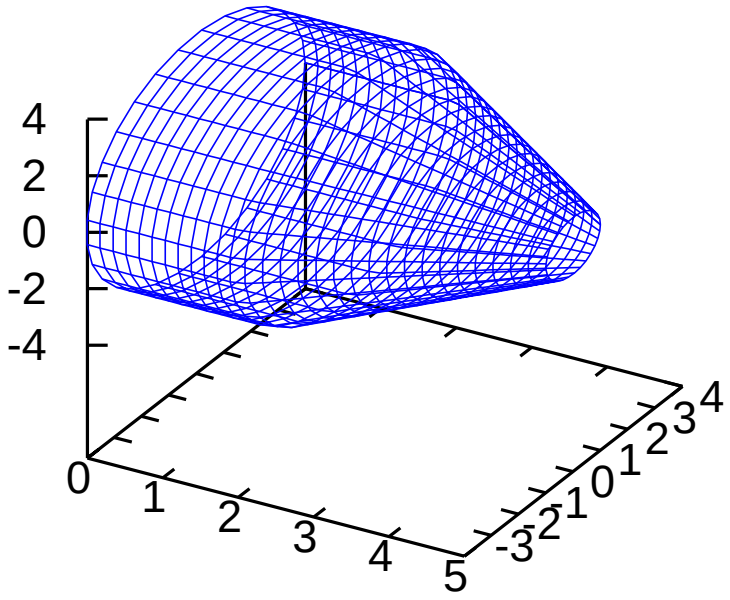
Die Flasche lässt sich durch folgende Funktion darstellen:

$$f(x) = \begin{cases} 4, \text{ für } x < 2 \\ -(x-2)+4 \end{cases}$$

mit $x \in [0;5]$. 5 E ist die Flasche hoch, unten hat sie 4 E als Radius.

```
(%i1) load (draw)$
(%i2) f(x) := if (x < 2) then
          4
        else
          -(x-2) + 4
        $
(%i3) draw3d (
    file_name = "flasche",
    terminal = jpg,
    dimensions = [500, 500],
    ztics = [-4, 2, 4],
    tube (
      x, 0, 0,
      f(x),
      x, 0, 5
    )
  )$
```

Abbildung 13.17: Der Code ist hier: 358



Sie definieren die Funktion als eine stückweise zusammengesetzte Funktion.

tube liefert das gewünschte. **tube** zeichnet Kreise um einen vorgegebenen Mittelpunkt mit einem vorgegebenen Radius. Die ersten drei Argumente von **tube** geben den Mittelpunkt an, das 4. Argument den Kreisradius und die letzten Argumente geben den Parameter und das Intervall des Parameters an.

tube hat folgende Argumente:

1. Die Funktion für die x-Komponente der Mittelpunkte der Kreise: Diese ist gleich x. Die Mittelpunkte der Kreise durchlaufen die x-Achse.

2. Die Funktion für die y-Komponente der Mittelpunkte der Kreise: 0. Die Kreise werden alle um die x-Achse gezeichnet.

3. Die Funktion für die z-Komponente der Mittelpunkte der Kreise: 0. Die Kreise werden alle um die x-Achse gezeichnet.

4. Der Radius am gegebenen Ort wird gerade durch $f(x)$ beschrieben.

5. x ist der Parameter

6. Die letzten beiden Werte geben das Intervall des Parameters an. Dieses ist gerade von 0 bis zur Höhe der Flasche: 5.

13.13 Die Höhenschnittpunkt verschiedener Dreiecke erkunden

Sie geben von einem Dreieck A und B vor und wollen C auf einer Parallelen verändern lassen. Die Punkte: A(0, 0) und B(1, 0). C befindet sich auf einer Parallelen zu AB welche den Abstand 1 zu AB hat.

Vergleichen Sie auch mit dem Problem: 8.10 S. 91.
Die Höhe hc ergibt sich dann wie folgt:

$$hc = C + r \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Die Normale zu b (AC) ist nb. Berechnen Sie C-A und vertauschen Sie die beiden Elemente und multiplizieren Sie einen mit (-1):

$$nb = \begin{pmatrix} C_y - A_y \\ -1(C_x - A_x) \end{pmatrix}$$

Die Höhe hb steht senkrecht auf AC und geht durch den Punkt B:

$$hb = B + s \cdot nb$$

Der Höhenschnittpunkt ist gerade der Schnittpunkt der beiden Geraden.

```
load (draw)$
A : [0, 0]$
B : [1, 0]$
dreieck(n) :=
  block ([],
    C : [n/10, 1],
    hc : [ C[1], C[2] + r ],
    nb : [(C - A )[2], (-C + A)[1]],
    hb : B + s * nb,
    lsg : solve ([hb[1] = hc[1], hb[2] = hc[2]], [r, s]),
    H : ev (hc, lsg),
    l : [points ([A, B]), points ([B, C]), points ([A, C])],
    l : append (l, [points ([A, H]), points ([B, H]), points ([C, H])]),
    return (l)
  )$
with_slider_draw (n, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    xrange = [-1, 2],
    yrange = [-1, 1.5],
    points_joined = true,
    dreieck (n)
    )$
```

Abbildung 13.18: Der Code ist hier: 359

Laden Sie den Code in ein Fenster von wxMaxima. Da hier ein Schieberegler verwendet wird, können Sie dies nur mit wxMaxima ausführen.

lappend (*Liste1, Liste2*) hängt Liste2 an Liste1. Die beiden Listen bleiben unverändert und eine kombinierte Liste wird zurückgegeben.

with_slider_draw (*Var, Liste mit Werte für Var, Graphik-Anweisungen*)

Wenn Sie das Bild anklicken, können Sie mit dem Schieberegler C verschieben.

13.14 Zeichnen einer Ebene

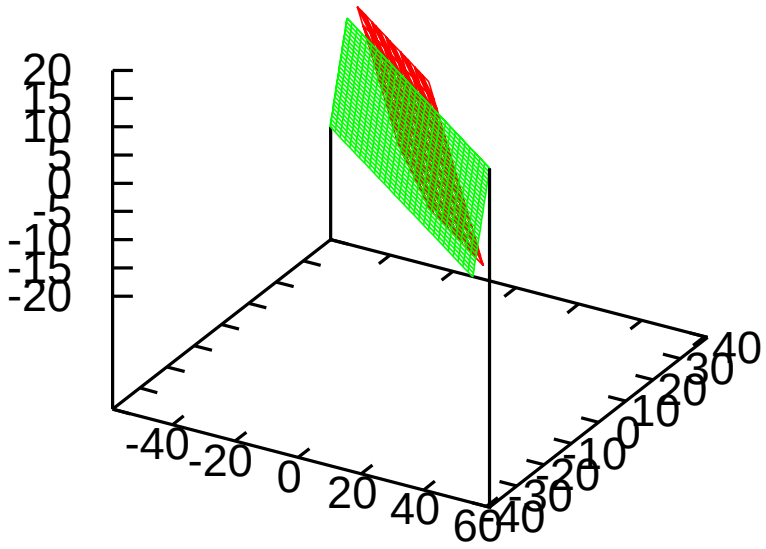
Sie haben eine Ebene in Koordinatenform oder Parameterform und möchten diese zeichnen lassen.

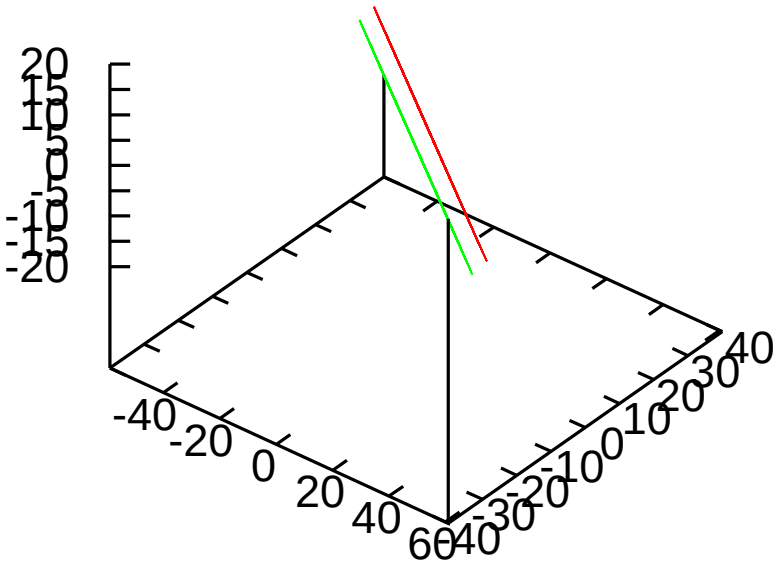
$$E : \vec{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + r \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix} + s \begin{pmatrix} -2 \\ 1 \\ 0 \end{pmatrix}$$

$$E : x_1 + 2 \cdot x_2 + x_3 = 5$$

```
(%i1) load (draw)$
(%i2) bild : [
    terminal = jpg,
    dimensions = [500, 500],
    color = red,
    implicit (x + 2*y + z = 10, x, -20, 20, y, -20, 20, z, -20, 20),
    color = green,
    parametric_surface (
        1 + r - 2*s,
        -r + s,
        r,
        r, -20 , 20,
        s, -20, 20)
    ]$
(%i3) draw3d (
    bild,
    file_name = "ebene"
)$
(%i4) draw3d (
    bild,
    file_name = "ebeneview",
    user_preamble = ["set view 51, 39"]
)$
```

Abbildung 13.19: Der Code ist hier: 360





implicit (*Funktion, x, xmin, xmax, y, ymin, ymax, z, zmin, zmax*) ist eine Option eines Graphikobjektes (gr2d

oder gr3d). Dort können Sie Ebenengleichung der Parameterform eingeben.

parametric_surface (*x-Wert, y-Wert, z-Wert, 1. Parameter, Minimum, Maximum, 2. Parameter, Minimum, Maximum*
ist ebenfalls eine Option eines Graphikobjektes (gr2d oder gr3d). Hier können Sie die Ebenengleichung der Koordinatenform eingeben.

In Gnuplot können Sie die Blickrichtung mit view angeben. Im zweiten Bild ist die Blickrichtung so gewählt, dass beide Ebenen im Bild parallel liegen und Sie genau auf die Kante schauen.

Wenn Sie das Standard-Gnuplot-Fenster als Ausgabe nehmen, können Sie mit der Maus die Blickrichtung ändern und das Objekt drehen.

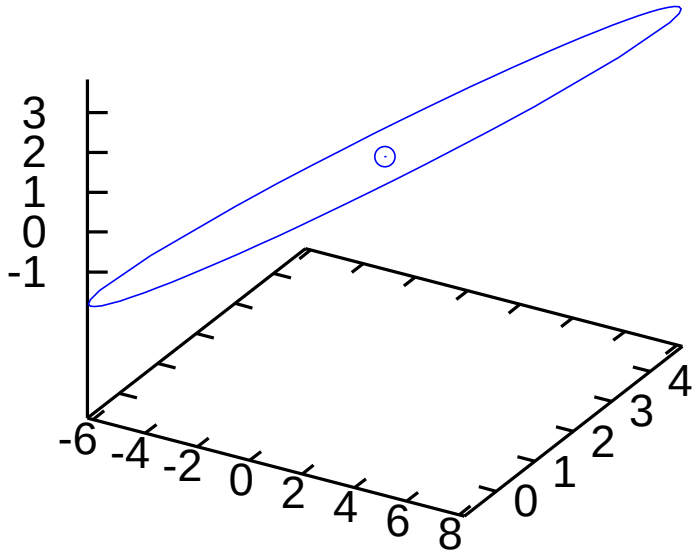
13.15 Zeichnen eines Kreises in einer Ebene

Sie haben eine Ebene vorgegeben:

$$E : \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} + t \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + s \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

In diese Ebene wollen Sie einen Ring mit dem Radius (r=2) um den Punkt p (1 / 2 / 1) zeichnen lassen.

```
(%i1) load (draw)$
(%i2) load (eigen)$
(%i3) v1 : columnvector ( [3, 1, 1] )$
(%i4) v2 : columnvector ( [2, 1, 1] )$
(%i5) p : columnvector ( [1, 2, 1] ) $
(%i6) r : 2$
(%i7) n : args (orthogonal_complement (v1, v2))[1];
0 errors, 0 warnings
                                  [  0  ]
                                  [     ]
(%o7)                             [ - 1 ]
                                  [     ]
                                  [  1  ]
(%i8) M : addcol (v1, v2, n);
                                [ 3  2   0  ]
                                [           ]
(%o8)                           [ 1  1  - 1 ]
                                [           ]
                                [ 1  1   1  ]
(%i9) kurve1 : columnvector ([u,  sqrt (r^2 - u^2), 0])$
(%i10) kurve2 : columnvector ([u, -sqrt (r^2 - u^2), 0])$
(%i11) ring1 : M . kurve1 + p;
                         [             2           ]
                         [ 2 sqrt(4 - u ) + 3 u + 1 ]
                         [                         ]
(%o11)                   [             2           ]
                         [   sqrt(4 - u ) + u + 2  ]
                         [                         ]
                         [             2           ]
                         [   sqrt(4 - u ) + u + 1  ]
(%i12) ring2 : M . kurve2 + p;
                        [              2           ]
                        [ - 2 sqrt(4 - u ) + 3 u + 1 ]
                        [                          ]
(%o12)                  [              2           ]
                        [   - sqrt(4 - u ) + u + 2  ]
                        [                          ]
                        [              2           ]
                        [   - sqrt(4 - u ) + u + 1  ]
(%i13) [x1, y1, z1] : [ring1[1][1], ring1[2][1], ring1[3][1]];
                     2                          2                    2
(%o13) [2 sqrt(4 - u ) + 3 u + 1, sqrt(4 - u ) + u + 2, sqrt(4 - u ) + u + 1]
(%i14) [x2, y2, z2] : args (map (lambda ([x], x[1]), ring2));
                      2                           2
(%o14) [- 2 sqrt(4 - u ) + 3 u + 1, - sqrt(4 - u ) + u + 2,
                                                                  2
                                                    - sqrt(4 - u ) + u +
                                                      1]
(%i15) draw3d (
    parametric (
       x1, y1, z1,
        u, -2, 2
```



load (*draw*) lädt das Paket zum zeichnen, **load** (*eigen*) das Paket, um **columnvector** (*Liste*) verwenden zu können.

Zuerst werden ein Punkt der Ebene und die Richtungsvektoren angegeben.

Mit **orthogonal_complement** (*columnvector, columnvector*) erhalten Sie einen Normalenvektor zu den beiden Richtungsvektoren. **args** (*Ausdruck*) entfernt **span**. span steht für alle Vielfache des Vektors.

Dann wird mit **addcol** (*Liste, . . .*) eine Matrix aus den Richtungsvektoren und dem Normalenvektor erstellt. Dies ist die Matrix, welche die Einheitsvektoren auf die Richtungsvektoren der Ebene, bzw den Normalenvektor der Ebene abbildet.

kurve1 und kurve2 ergeben einen Kreis mit dem Radius r in der x-y-Ebene um den Nullpunkt.

Durch die Multiplikation von der Matrix M mit der kurve1 und anschliessender Verschiebung um p erhalten Sie den Ring in der Ebene.

In x1, y1 und z1 werden die Koordinaten in Abhängigkeit eines Parameters u angegeben. Sie können diese Werte auf zwei verschiedene Arten übergeben. (%i13) und (%i14) machen im Prinzip dasselbe.

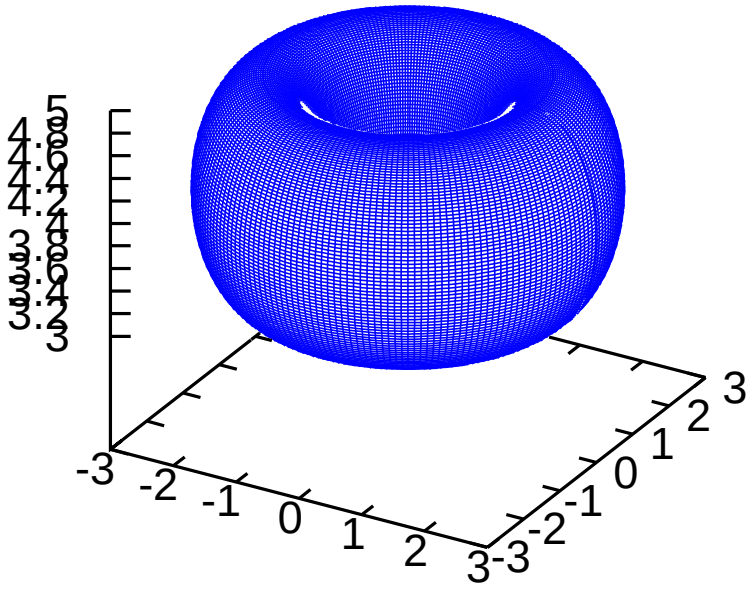
draw3d (*Graphikobjekt*) zeichnet diese beiden Halbkreise.

13.16 Torus zeichnen

Sie wollen einen Torus zeichnen. Der Mittelpunkt der Kreise soll 2 betragen. Die Radien der Kreise sind 1. Der Torus soll in der Höhe von 4 über der x-y-Ebene schweben.

```
(%i1) load (draw)$
(%i2) r : 2; /* Radius des Mittelpunktes der Kreisringe des Torus */
(%o2)                                  2
(%i3) draw3d (
(%i3)       file_name = "torus",
(%i3)       terminal = jpg,
(%i3)       dimensions = [500, 500],
(%i3)       surface_hide = true,
(%i3)       xu_grid = 200, /* Anzahl der x Punkte */
(%i3)       yv_grid = 200, /* Anzahl der y Punkte */
(%i3)       tube (
(%i3)         r * cos (u), /* x-Koord. */
(%i3)         r * sin (u), /* y-Koord. */
(%i3)         4,           /* z-Koord. */
(%i3)         1,           /* Innenradius des Torus */
(%i3)         u,           /* Parameter */
(%i3)         0,           /* Intervallstart fuer u */
(%i3)         2*%pi        /* Intervallende fuer u */
(%i3)       )
(%i3)     )$
```

Abbildung 13.21: Der Code ist hier: 362



tube (*xfunktion, yfunktion, zfunktion, radiusfunktion, parameter, pmin, pmax*). pmin und pmax sind die Intervallgrenzen des Parameters. parameter ist der Buchstabe des Parameters.

tube hat mehrere Optionen:

- xu_grid – Anzahl der x Punkte
- yv_grid – Anzahl der y Punkte
- line_type – solid (durchgezogene Linie) oder dots (Punkte)
- line_width – Liniendicke
- color – Farbe
- key – Title
- enhanced3d – Farbpalette erstellen
- surface_hide – true: versteckt nicht sichtbare Flächen
- transform

Damit der Torus „schön“ wird, müssen Sie xu_grid und yv_grid erhöhen.

tube zeichnet Kreise, an der jeweiligen Position mit einem gegebenen Radius. Sie benötigen also nicht eine Parameterdarstellung eines Torus.

13.17 Einstellungen der Graphik

In diesem Abschnitt werden Ihnen die verschiedenen Einstellmöglichkeiten der Farben, Linien usw. gezeigt.

13.17.1 Wie stellt man Farben ein

Sie möchten Graphiken, bzw. Beschriftungen farbig gestalten.

```
(%i1) Reihe1 : [
    color = black,           label (["black",          -50, 100]),
    color = dark-grey,       label (["dark-grey",      -50, 95]),
    color = red,             label (["red",            -50, 90]),
    color = "#00c000",       label (["web-green",      -50, 85]),
    color = "#0080ff",       label (["web-blue",       -50, 80]),
    color = dark-magenta,    label (["dark-magenta",   -50, 75]),
    color = dark-cyan,       label (["dark-cyan",      -50, 70]),
    color = dark-orange,     label (["dark-orange",    -50, 65]),
    color = dark-yellow,     label (["dark-yellow",    -50, 60]),
    color = royalblue,       label (["royalblue",      -50, 55]),
    color = goldenrod,       label (["goldenrod",      -50, 50]),
    color = "#008040",       label (["dark-spring-green", -50, 45]),
    color = purple,          label (["purple",         -50, 40]),
    color = "#306080",       label (["steelblue",      -50, 35]),
    color = dark-red,        label (["dark-red",       -50, 30]),
    color = "#408000",       label (["dark-chartreuse", -50, 25]),
    color = "#ff80ff",       label (["orchid",         -50, 20]),
    color = aquamarine,      label (["aquamarine",     -50, 15]),
    color = brown,           label (["brown",          -50, 10]),
    color = yellow,          label (["yellow",         -50, 5]) ]$
(%i2) Reihe2 : [
    color = turquoise,       label (["turquoise",      -20, 100]),
    color = grey,            label (["grey",           -20, 95]),
    color = grey0,           label (["grey0",          -20, 90]),
    color = grey10,          label (["grey10",         -20, 85]),
    color = grey20,          label (["grey20",         -20, 80]),
    color = grey30,          label (["grey30",         -20, 70]),
    color = grey40,          label (["grey40",         -20, 75]),
    color = grey50,          label (["grey50",         -20, 60]),
    color = grey60,          label (["grey60",         -20, 65]),
    color = grey70,          label (["grey70",         -20, 50]),
    color = grey80,          label (["grey80",         -20, 55]),
    color = grey90,          label (["grey90",         -20, 40]),
    color = grey100,         label (["grey100",        -20, 45]),
    color = light-red,       label (["light-red",      -20, 30]),
    color = light-green,     label (["light-green",    -20, 25]),
    color = light-blue,      label (["light-blue",     -20, 20]),
    color = light-magenta,   label (["light-magenta",  -20, 15]),
    color = light-cyan,      label (["light-cyan",     -20, 10]),
    color = light-goldenrod, label (["light-goldenrod", -20, 5]),
    color = light-pink,      label (["light-pink",     -20, 0]) ]$
```

Abbildung 13.22: Der Code ist hier: 363

```
(%i1) Reihe3 : [
    color = light-turquoise, label (["light-turquoise", 10, 100]),
    color = gold,            label (["gold",          10, 95]),
    color = green,           label (["green",         10, 90]),
    color = dark-green,      label (["dark-green",    10, 85]),
    color = spring-green,    label (["spring-green",  10, 80]),
    color = forest-green,    label (["forest-green",  10, 75]),
    color = sea-green,       label (["sea-green",     10, 70]),
    color = blue,            label (["blue",          10, 65]),
    color = dark-blue,       label (["dark-blue",     10, 60]),
    color = midnight-blue,   label (["midnight-blue", 10, 55]),
    color = navy,            label (["navy",          10, 50]),
    color = medium-blue,     label (["medium-blue",   10, 45]),
    color = skyblue,         label (["skyblue",       10, 40]),
    color = cyan,            label (["cyan",          10, 35]),
    color = magenta,         label (["magenta",       10, 30]),
    color = dark-turquoise,  label (["dark-turquoise", 10, 25]),
    color = dark-pink,       label (["dark-pink",     10, 20]),
    color = coral,           label (["coral",         10, 15]),
    color = light-coral,     label (["light-coral",   10, 10]),
    color = orange-red,      label (["orange-red",    10, 5]) ]$
(%i2) Reihe4 : [
    color = salmon,          label (["salmon",        40, 100]),
    color = dark-salmon,     label (["dark-salmon",   40, 95]),
    color = khaki,           label (["khaki",         40, 90]),
    color = dark-khaki,      label (["dark-khaki",    40, 85]),
    color = dark-goldenrod,  label (["dark-goldenrod", 40, 80]),
    color = beige,           label (["beige",         40, 75]),
    color = "#a08020",       label (["olive",         40, 70]),
    color = violet,          label (["violet",        40, 65]),
    color = dark-violet,     label (["dark-violet",   40, 60]),
    color = plum,            label (["plum",          40, 55]),
    color = "#905040",       label (["dark-plum",     40, 50]),
    color = "#556b2f",       label (["dark-olivegreen", 40, 45]),
    color = "#801400",       label (["orangered4",    40, 40]),
    color = "#801414",       label (["brown4",        40, 35]),
    color = "#804014",       label (["sienna4",       40, 30]),
    color = "#804080",       label (["orchid4",       40, 25]),
    color = "#8060c0",       label (["mediumpurple3", 40, 20]),
    color = "#8060ff",       label (["slateblue1",    40, 15]),
    color = "#808000",       label (["yellow4",       40, 10]),
    color = "#ff8040",       label (["sienna1",       40, 5]) ]$
```

Abbildung 13.23: Der Code ist hier: 364

```
(%i1) Reihe5 : [
    color = "#ffa040",        label (["tan1",          70, 100]),
    color = "#ffa060",        label (["sandybrown",    70, 95]),
    color = light-salmon,     label (["light-salmon",  70, 90]),
    color = pink,             label (["pink",          70, 85]),
    color = "#ffff80",        label (["khaki1",        70, 80]),
    color = "#ffffc0",        label (["lemonchiffon",  70, 75]),
    color = "#cdb79e",        label (["bisque",        70, 70]),
    color = "#f0fff0",        label (["honeydew",      70, 65]),
    color = "#a0b6cd",        label (["slategrey",     70, 60]),
    color = "#c1ffc1",        label (["seagreen",      70, 55]),
    color = "#cdc0b0",        label (["antiquewhite",  70, 50]),
    color = "#7cff40",        label (["chartreuse",    70, 45]),
    color = "#a0ff20",        label (["greenyellow",   70, 40]),
    color = gray,             label (["gray",          70, 35]),
    color = light-gray,       label (["light-gray",    70, 30]),
    color = light-grey,       label (["light-grey",    70, 25]),
    color = dark-gray,        label (["dark-gray",     70, 20]),
    color = "#a0b6cd",        label (["slategray",     70, 15]),
    color = gray0,            label (["gray0",         70, 10]),
    color = gray10,           label (["gray10",        70, 5]) ]$
(%i2) Reihe6: [
    color = gray20,           label (["gray20",        100, 100]),
    color = gray30,           label (["gray30",        100, 95]),
    color = gray40,           label (["gray40",        100, 90]),
    color = gray50,           label (["gray50",        100, 85]),
    color = gray60,           label (["gray60",        100, 80]),
    color = gray70,           label (["gray70",        100, 75]),
    color = gray80,           label (["gray80",        100, 70]),
    color = gray90,           label (["gray90",        100, 65]),
    color = gray100,          label (["gray100",       100, 60]) ]$
```

Abbildung 13.24: Der Code ist hier: 365

```
(%i1) load (draw)$
(%i2) batchload ("GraphikEinstellungenColor1.mac")$
(%i3) batchload ("GraphikEinstellungenColor2.mac")$
(%i4) batchload ("GraphikEinstellungenColor3.mac")$
(%i5) einstellungen : [
    user_preamble = ["set noborder"],
    xtics = false,
    ytics = false,
    dimensions = [800, 600],
    xrange = [-60, 120],
    yrange = [0, 120],
    terminal = jpg
]$
(%i6) bild1 : gr2d (
    background_color = "#ffffff",
    file_name = "color",
    einstellungen,
    Reihe1, Reihe2, Reihe3, Reihe4, Reihe5, Reihe6
)$
(%i7) bild2 : gr2d (
    data_file_name = "data2.gnuplot",
    gnuplot_file_name = "maxout2.gnuplot",
    background_color = "#000000",
    file_name = "color_black",
    einstellungen,
    Reihe1, Reihe2, Reihe3, Reihe4, Reihe5, Reihe6
)$
(%i8) draw (bild1)$
(%i9) draw (bild2)$
```

Abbildung 13.25: Der Code ist hier: 366

Die Reihen sind in den Dateien „GraphikEinstellungenColor1.mac“ usw. abgespeichert. Die Farbbezeichnungen sind der Gnuplot-Dokumentation entnommen.

Es werden zwei Bilder geschrieben. Das eine Bild hat einen weißen und das andere einen schwarzen Hintergrund.

Die Option **background_color** färbt den Hintergrund. Sie können die normalen Ausgabefenster färben lassen und bei Dateien nur gif, png jpg und gif Dateien.

Da hier zwei Bilder gezeichnet werden und dabei temporäre Dateien angelegt werden, müssen Sie unter Umständen eine kurze Pause zwischen den beiden **draw** (*Graphikobjekt*) Befehlen einlegen. Ansonsten bekommen Sie evtl. Fehlermeldungen. Eine andere Möglichkeit ist, wie sie hier praktiziert wurde, für das 2. Bild andere temporäre Dateien anzulegen, in denen die numerischen Daten für Gnuplot abgespeichert werden.

In Gnuplot gibt es 122 vordefinierte Farbnamen:

white	#ffffff	=	255	255	255
black	#000000	=	0	0	0
dark-grey	#a0a0a0	=	160	160	160
red	#ff0000	=	255	0	0
web-green	#00c000	=	0	192	0
web-blue	#0080ff	=	0	128	255
dark-magenta	#c000ff	=	192	0	255
dark-cyan	#00eeee	=	0	238	238
dark-orange	#c04000	=	192	64	0
dark-yellow	#c8c800	=	200	200	0
royalblue	#4169e1	=	65	105	225
goldenrod	#ffc020	=	255	192	32
dark-spring-green	#008040	=	0	128	64
purple	#c080ff	=	192	128	255
steelblue	#306080	=	48	96	128
dark-red	#8b0000	=	139	0	0
dark-chartreuse	#408000	=	64	128	0
orchid	#ff80ff	=	255	128	255
aquamarine	#7fffd4	=	127	255	212
brown	#a52a2a	=	165	42	42
yellow	#ffff00	=	255	255	0
turquoise	#40e0d0	=	64	224	208
grey0	#000000	=	0	0	0
grey10	#1a1a1a	=	26	26	26
grey20	#333333	=	51	51	51
grey30	#4d4d4d	=	77	77	77
grey40	#666666	=	102	102	102
grey50	#7f7f7f	=	127	127	127
grey60	#999999	=	153	153	153
grey70	#b3b3b3	=	179	179	179
grey	#c0c0c0	=	192	192	192
grey80	#cccccc	=	204	204	204
grey90	#e5e5e5	=	229	229	229
grey100	#ffffff	=	255	255	255
light-red	#f03232	=	240	50	50
light-green	#90ee90	=	144	238	144
light-blue	#add8e6	=	173	216	230
light-magenta	#f055f0	=	240	85	240
light-cyan	#e0ffff	=	224	255	255
light-goldenrod	#eedd82	=	238	221	130
light-pink	#ffb6c1	=	255	182	193
light-turquoise	#afeeee	=	175	238	238
gold	#ffd700	=	255	215	0
green	#00ff00	=	0	255	0
dark-green	#006400	=	0	100	0
spring-green	#00ff7f	=	0	255	127
forest-green	#228b22	=	34	139	34
sea-green	#2e8b57	=	46	139	87

blue	#0000ff	=	0	0	255
dark-blue	#00008b	=	0	0	139
midnight-blue	#191970	=	25	25	112
navy	#000080	=	0	0	128
medium-blue	#0000cd	=	0	0	205
skyblue	#87ceeb	=	135	206	235
cyan	#00ffff	=	0	255	255
magenta	#ff00ff	=	255	0	255
dark-turquoise	#00ced1	=	0	206	209
dark-pink	#ff1493	=	255	20	147
coral	#ff7f50	=	255	127	80
light-coral	#f08080	=	240	128	128
orange-red	#ff4500	=	255	69	0
salmon	#fa8072	=	250	128	114
dark-salmon	#e9967a	=	233	150	122
khaki	#f0e68c	=	240	230	140
dark-khaki	#bdb76b	=	189	183	107
dark-goldenrod	#b8860b	=	184	134	11
beige	#f5f5dc	=	245	245	220
olive	#a08020	=	160	128	32
orange	#ffa500	=	255	165	0
violet	#ee82ee	=	238	130	238
dark-violet	#9400d3	=	148	0	211
plum	#dda0dd	=	221	160	221
dark-plum	#905040	=	144	80	64
dark-olivegreen	#556b2f	=	85	107	47
orangered4	#801400	=	128	20	0
brown4	#801414	=	128	20	20
sienna4	#804014	=	128	64	20
orchid4	#804080	=	128	64	128
mediumpurple3	#8060c0	=	128	96	192
slateblue1	#8060ff	=	128	96	255
yellow4	#808000	=	128	128	0
sienna1	#ff8040	=	255	128	64
tan1	#ffa040	=	255	160	64
sandybrown	#ffa060	=	255	160	96
light-salmon	#ffa070	=	255	160	112
pink	#ffc0c0	=	255	192	192
khaki1	#ffff80	=	255	255	128
lemonchiffon	#ffffc0	=	255	255	192
bisque	#cdb79e	=	205	183	158
honeydew	#f0fff0	=	240	255	240
slategrey	#a0b6cd	=	160	182	205
seagreen	#c1ffc1	=	193	255	193
antiquewhite	#cdc0b0	=	205	192	176
chartreuse	#7cff40	=	124	255	64
greenyellow	#a0ff20	=	160	255	32

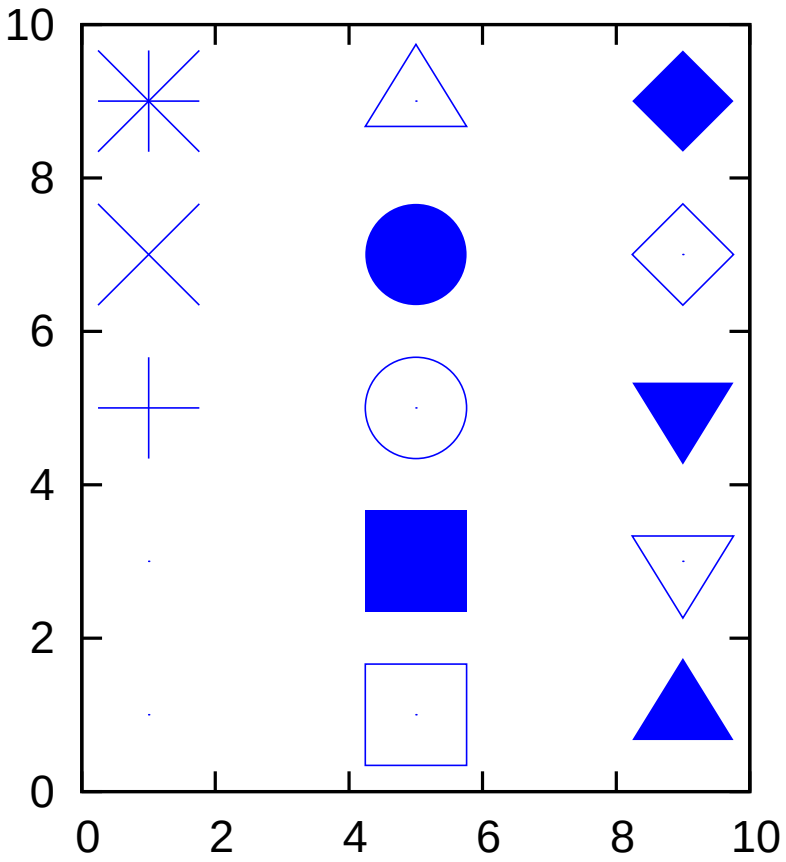
gray	#bebebe	=	190	190	190
light-gray	#d3d3d3	=	211	211	211
light-grey	#d3d3d3	=	211	211	211
dark-gray	#a0a0a0	=	160	160	160
slategray	#a0b6cd	=	160	182	205
gray0	#000000	=	0	0	0
gray10	#1a1a1a	=	26	26	26
gray20	#333333	=	51	51	51
gray30	#4d4d4d	=	77	77	77
gray40	#666666	=	102	102	102
gray50	#7f7f7f	=	127	127	127
gray60	#999999	=	153	153	153
gray70	#b3b3b3	=	179	179	179
gray80	#cccccc	=	204	204	204
gray90	#e5e5e5	=	229	229	229
gray100	#ffffff	=	255	255	255

13.17.2 Verändern der Punkte

Welche Punkte kann man benutzen und wie sieht es aus?

```
(%i1) load (draw)$
(%i2) draw2d(
    xrange = [0,10],
    yrange = [0,10],
    point_size = 5,
    file_name = "points",
    terminal = jpg,
    dimensions = [500, 500],
    point_type =  -1, points ([[1, 1]]),
    point_type =  0, points ([[1, 3]]),
    point_type =  1, points ([[1, 5]]),
    point_type =  2, points ([[1, 7]]),
    point_type =  3, points ([[1, 9]]),
    point_type =  4, points ([[5, 1]]),
    point_type =  5, points ([[5, 3]]),
    point_type =  6, points ([[5, 5]]),
    point_type =  7, points ([[5, 7]]),
    point_type =  8, points ([[5, 9]]),
    point_type =  9, points ([[9, 1]]),
    point_type =  10, points ([[9, 3]]),
    point_type =  11, points ([[9, 5]]),
    point_type =  12, points ([[9, 7]]),
    point_type =  13, points ([[9, 9]])
    )$
```

Abbildung 13.26: Der Code ist hier: 367



Sie verändern mit **point_type** die Punktewahl.
Beachten Sie, dass Sie zwei eckige Klammern bei points benutzen müssen.

13.18 Optionen von implicit

Sie möchten bei draw2d oder draw3d Funktionen implizit beschreiben mit **implicit**.
Sie haben folgende Einstellungsmöglichkeiten:

- **enhanced3d** aber nur das [f(x,y,z), x, y, z] Modell.
- **ip_grid** default: [50, 50].
- **ip_grid_in** default: [5, 5].
- **key** Name der Funktion in der Legende.
- **line_width** default: 1.
- **x_voxel** default: 10.
- **y_voxel** default: 10.
- **color**

13.18.1 Einstellungen des Intervalles auf der y-Achse

Sie wollen eine Funktion in einem bestimmten Intervall für die y-Werte zeichnen lassen.
Dazu gibt es verschiedene Lösungen:

1. Sie geben bei draw2d als Option an:
   yrange = [0, 10]
   Dann werden nur die y-Werte zwischen 0 und 10 gezeichnet.

2. Sie benutzen ein Gnuplot Kommando. Dazu geben Sie bei draw2d als Option an:
   user_preamble = [“set yrange [0: 10]“]

3. Wenn Sie die nur die Werte ab 0 zeichnen lassen wollen und die anderen Werte automatisch berechnen lassen wollen, so geht dies mit dem Gnuplot Kommando:
   user_preamble = [“set yrange [0:]“]
   Sie lassen die obere Intervallsgrenze einfach weg. Entsprechend können Sie auch nur eine obere Intervallsgrenze angeben.

13.19 Graphikelemente von Draw

Sie wollen vordefinierte Grafikobjekte benutzen.

13.19.1 2-dim Objekte für draw2d

1. Säulen für ein Säulendiagramm: bars

2. Kreise und Ellipsen: ellipse
   Für einen Kreis müssen Sie die Halbachsen a und b gleich groß wählen.

3. Pfeile: vector
   Für die Pfeilgröße werden Sie voraussichtlich head_length anpassen müssen.

4. Punkte: points

5. Dreieck: triangle

6. Rechteck: rectangle

7. Viereck: quadrilateral

8. Polygon: polygon

13.19.2 3-dim Objekte für draw3d

1. Pfeil: vector mit head_length

2. Kugel, Halbkugel: spherical

3. Zylinder (auch verformter Zylinder): tube, bzw tube_extremes, cylindrical

4. Dreieck: triangle

5. Viereck im Raum: quadrilateral

6. In Zylinderkoordinaten zeichnen: cylindrical
7. In Kugelkoordinaten zeichnen: spherical

Kapitel 14

Zahlentheorie

14.1 Bestimmen des kleinsten gemeinsamen Vielfachen

Sie wollen das kleinste gemeinsame Vielfache von mehreren Zahlen bestimmen

```
(%i1) load (functs)$
(%i2) lcm (2, 4, 6);
(%o2)                                12
(%i3) g : [2, 4, 6];
(%o3)                             [2, 4, 6]
(%i4) lcm (g);
(%o4)                                12
(%i5) lcm (x^2, x);
                                      2
(%o5)                                 x
```

Abbildung 14.1: Der Code ist hier: 368

Sie müssen zuerst mit **load** (*func*) den Befehl **lcm** bereitstellen.

lcm (*Ausdruck 1, . . . , Ausdruck n*) bestimmt das kleinste gemeinsame Vielfache dieser Ausdrucke. Dies können Zahlen sein, Polynome oder auch Listen.

14.2 Bestimmen des größten gemeinsamen Teilers

Sie wollen den größten gemeinsamen Teiler von zwei Zahlen bestimmen.

```
(%i1) gcd (12, 16);
(%o1)                                4
(%i2) p1 : (x-1) * (x+2);
(%o2)                          (x - 1) (x + 2)
(%i3) p2 : (x-1) * (x+3);
(%o3)                          (x - 1) (x + 3)
(%i4) gcd (p1, p2);
(%o4)                              x - 1
```

Abbildung 14.2: Der Code ist hier: 369

gcd (*ZahlentheorieGGV.mac*)

Beachten Sie, dass Sie nur zwei Zahlen und auch keine Liste übergeben dürfen. Sie können aber auch den ggT von Polynomen bilden.

14.3 Ist eine Zahl eine Primzahl?

Sie wollen von einer Zahl wissen, ob sie eine Primzahl ist.

```
(%i1) primep (12);
(%o1)                              false
(%i2) primep (7);
(%o2)                              true
```

Abbildung 14.3: Der Code ist hier: 370

Der Befehl **primep** (*ganze Zahl*) testet, ob eine Zahl eine Primzahl ist oder nicht. Ist Sie eine Primzahl, wird true zurückgeliefert, sonst false. Der Primzahltest beachtet nicht das Vorzeichen. **primep** (-7) ist also auch true.

14.4 Ausgabe der ersten 10 Primzahlen

Sie wollen eine Liste der Primzahlen ausgeben.

```
(%i1) a : 1$
(%i2) for i thru 10 do
    print ( a : next_prime (a) )
  $
2
3
5
7
11
13
17
19
23
29
(%i3) prim : primes (0, 50);
(%o3)      [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
(%i4) rest ( prim, 10 - length (prim) );
(%o4)                 [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Abbildung 14.4: Der Code ist hier: 371

Sie weisen der Variablen a die Zahl 1 zu. dann wird die **for** (*Anweisungen*) - Schleife 10 mal durchlaufen.

next_prime (*Zahl*) gibt die nächstgrößere Primzahl aus. Diese weisen Sie a zu und lassen dann das a ausdrucken mit **print** (*Ausdruck*)

Primzahlen zwischen a und b (einschließlich) können Sie sich mit **primes** (*a, b*) ausgeben lassen. Alle Primzahlen, welche kleiner als 50 sind erhalten Sie also mit **primes** (*0, 50*).

rest (*expr, n*) bearbeitet den Ausdruck expr aufgrund des Vorzeichens von n unterschiedlich.

1. n ist positiv.
   Dann werden die ersten n Elemente von expr gelöscht.

2. n ist negativ.
   Dann werden die letzten n Elemente von expr gelöscht.

length (*Liste*) gibt Ihnen die Anzahl der Elemente der Liste. **length**(prim) gibt Ihnen die Anzahl der Primzahlen, welche kleiner als 50 sind.

Wenn Sie jetzt wissen wollen, wieviele Elemente gelöscht werden sollen, damit Sie 10 Elemente erhalten, dann müssen Sie von der Anzahl (length(prim)) 10 abziehen. Da Sie die letzten Elemente löschen wollen müssen Sie diesen Ausdruck mit (-1) multiplizieren.

14.5 Bestimmen der Teiler einer Zahl

Sie haben eine Zahl und wollen deren Teiler bestimmen.

```
(%i1) s : divisors (12);
(%o1)                          {1, 2, 3, 4, 6, 12}
(%i2) l : args (s);
(%o2)                          [1, 2, 3, 4, 6, 12]
(%i3) l[5];
(%o3)                                   6
```

Abbildung 14.5: Der Code ist hier: 372

Der Befehl **divisors** (*ganze Zahl*) gibt Ihnen die Teiler der Zahl. Die 1 und die Zahl selber werden ebenfalls zurückgeliefert.

Die Rückgabe erfolgt nicht als Liste sondern als Menge (set). Sie können auf die einzelnen Elemente nicht zugreifen wie auf die Elemente einer Liste.

Um das set in eine Menge umzuwandeln benutzen Sie **listify** (*set*). Auf die Liste können Sie wie gewohnt zugreifen.

14.6 Primzahlzerlegung

Sie wollen eine Zahl in Primzahlen zerlegen.

```
(%i1) r : ifactors (12);
(%o1)                             [[2, 2], [3, 1]]
(%i2) r : ifactors (12), factors_only : true;
(%o2)                                  [2, 3]
```

Abbildung 14.6: Der Code ist hier: 373

Lösen Sie das Problem mit **ifactors** (*positive ganze Zahl*)

Der Befehl **ifactors** (*positive ganze Zahl*) gibt Ihnen eine Liste, deren Elemente jeweils eine 2-elementige Liste sind. Das erste Element ist jeweils die Primzahl, das zweite Element ist die Häufigkeit bei der Primzahlzerlegung. Die gesuchte Liste der Primzahlen ist eine Liste, welche aus dem jeweils ersten Element aller dieser Listen besteht.

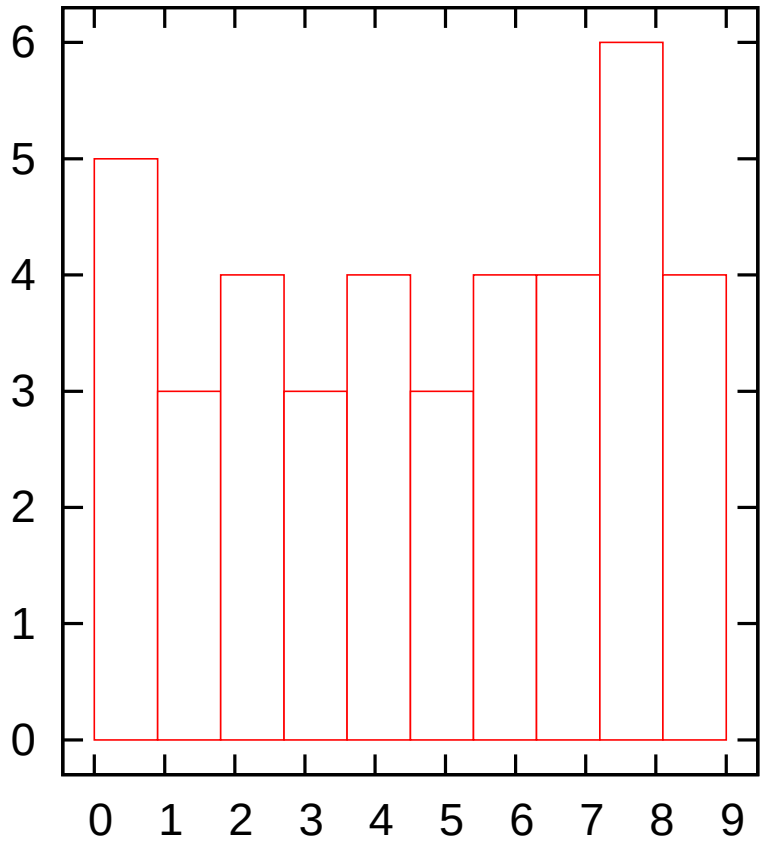
Die Option factors_only gibt Ihnen dann die gesuchte Liste der Primzahlen aus.

14.7 Erstellen eines Histogramms der Nachkommastellen einer reellen Zahl

Sie wollen das Histogramm der ersten 40 Zahlen der Nachkommastellen von Wurzel 2 erstellen.

```
(%i1) genauigkeit : 40$
(%i2) fpprec : genauigkeit$
(%i3) z : bfloat (sqrt(2));
(%o3)             1.4142135623730950488016887242096980785 7b0
(%i4) l : map (parse_string, charlist (substring (string (z), 3, fpprec )));
(%o4) [4, 1, 4, 2, 1, 3, 5, 6, 2, 3, 7, 3, 0, 9, 5, 0, 4, 8, 8, 0, 1, 6, 8, 8,
                                          7, 2, 4, 2, 0, 9, 6, 9, 8, 0, 7, 8,
5]
(%i5) load (descriptive)$
(%i6) histogram (
    l,
    nclasses = 10,
    file_name = "nachkommastellen",
    terminal = jpg,
    dimensions = [500, 500]
    )$
```

Abbildung 14.7: Der Code ist hier: 374



Mit **fpprec** legen Sie die Genauigkeit fest. Hier werden die Nachkommastellen bei bfloats auf 40 festgelegt. Die „normalen“ float-Zahlen behalten ihre Standardlänge von 16. Mit **fpprec** können Sie eine beliebige Genauigkeit einstellen.

z ist ein bfloat und hat enthält jetzt die Wurzel von 2 mit 40 Nachkommastellen.

Der folgende Befehl formt diese Zahl, welche in z gespeichert ist, in eine Zeichenkette um. Diese Zeichenkette kann dann so bearbeitet werden, dass jedes Zeichen als Element in einer Liste geschrieben ist. Dann müssen diese Zeichen anschließend in Zahlen umgewandelt werden. Maxima kann die 4 als Zeichen wie jeden Buchstaben des Alphabets gespeichert haben oder als Zahl. Wenn es ein Zeichen ist, dann kann Maxima nicht mit der „4“ rechnen, weil es im Prinzip denkt, dass es ein Buchstabe sei. Daher müssen Sie hier jeweils geschickt umwandeln.

Wenn Sie den Befehl selber auseinander nehmen und ausprobieren, sehen Sie den Unterschied, ob Zeichen oder Zahl, den Listen nicht an.

Den folgenden Befehl untersuchen wir in der Mitte beginnend.

1. **string** (*z*) formt die Zahl z in eine Zeichenkette um.

2. **substring** (*string(z), 3, fpprec*) fpprec haben wir oben die Zahl 40 zugewiesen. Damit können wir den Befehl umschreiben:
   substring (*1.414 . . . , 3, 40*)
   nimmt eine Zeichenkette aus 1.414. . . heraus. Die Funktion beginnt bei dem 3. Zeichen und nimmt dann die nächsten 40 Zeichen (also alle folgenden).

   Damit haben Sie nun folgende Zeichenkette!:
   4142135623730950488016887242096980785

3. Jedes dieser Zeichen schreiben Sie jetzt als Element in einer Liste mit **charlist** (*Zeichenkette*).

4. Von dieser Liste möchten Sie jetzt jedes Zeichen – also jedes Element – in eine Zahl umwandeln. Durchlaufen Sie diese Liste und wenden Sie auf jedes Element die Funktion **parse_string** (*Zeichen*) an. **parse_string**(“4“) macht aus einem Zeichen eine Zahl.

   map (*Funktion, Ausdruck, . . .*) durchläuft diese Zeichenkette und führt auf jeden Ausdruck die vorne stehende Funktion aus: **map** (parse_string, [4, 1, 3 . . .]) parse_string wird also auf jedes Listenelement angewendet und wandelt also alle Zeichen in Zahlen um.

   Sie erhalten eine Liste mit Zahlen! [4, 1, 4 . . .]

Mit **load**(desciptive) is es möglich den Befehl **histogram** (*Liste, Optionen*) aufzurufen.

histogram (*Liste, Optionen*) unterteilt durch die Option nclasses = 10 die Liste in 10 Bereiche. Da Sie auch 10 unterschiedliche Zahlen haben ergibt sich für jede Zahl auch eine Säule.

14.8 Die Binomischen Formeln

Sie wollen eine der Binomischen Formeln darstellen (oder eine höhere Potenz als zwei):

```
(%i1) expand ( (a+b)^2 );
                                2             2
(%o1)                          b  + 2 a b + a
(%i2) expand ( (a+b)^3 );
                             3         2       2       3
(%o2)                       b  + 3 a b  + 3 a  b + a
```

Abbildung 14.8: Der Code ist hier: 375

Sie geben die Formel ein und der Befehl **expand** (*Polynom*) multipliziert die Formel für Sie aus.

14.9 Erstellen der Fibonacci-Zahlen

Sie möchten die Fibonacci-Zahlen ausgeben.

Diese sind rekursiv definiert:

$$a_1 = 1$$
$$a_2 = 1$$
$$a_n = a_{n-1} + a_n n - 2$$

```
(%i1) a[1] : 1;
(%o1)                                1
(%i2) a[2] : 1;
(%o2)                                1
(%i3) a[n] := a[n-1] + a[n-2];
                              a  := a      + a
(%o3)                          n     n - 1    n - 2
(%i4) for i : 1 while i < 10 do
    print (a[i])$
1
1
2
3
5
8
13
21
34
```

Abbildung 14.9: Der Code ist hier: 376

Sie definieren die ersten Werte. Dann definieren Sie eine Funktion „:=“ rekursiv.
for (*Anweisungen*) gibt Ihnen mit einer Schleife die ersten 10 Zahlen der Reihe aus.
print (*Ausdrücke*) druckt die Werte.

Kapitel 15

Listen

Dieses Kapitel löst keine Probleme, welche Sie zu Maxima greifen lassen. Da aber Maxima intern mit Listen arbeitet sind Sie immer wieder dazu gezwungen Listen zu manipulieren.

Wenn Sie auf die Elemente einer Liste zugreifen wollen wird in der Regel der Befehl **map** (*Funktion, Liste*) angewendet.

ausgabeliste : map (f, eingabeliste);

eingabeliste ist eine Liste. ausgabeliste ist dann ebenfalls eine Liste, welche genau so vielen Elementen wie eingabeliste enthält.

Jedes Element von eingabeliste wird der Funktion f übergeben.

15.1 Elemente einer Liste

Sie wollen auf die Elemente einer Liste zugreifen.

```
(%i1) l : [1, 2, 3, 4, 5];
(%o1)                          [1, 2, 3, 4, 5]
(%i2) g : [Max, Erna, Paul, Wilma];
(%o2)                     [Max, Erna, Paul, Wilma]
(%i3) /* 1. Moeglichkeit */
l[2];
(%o3)                                  2
(%i4) g[2];
(%o4)                                 Erna
(%i5) /* 2. Moeglichkeit */
second (l);
(%o5)                                  2
(%i6) second (g);
(%o6)                                 Erna
```

Abbildung 15.1: Der Code ist hier: 377

Sie können entweder mit rechteckigen Klammern auf die Listenelemente zugreifen oder mit einem extra Befehl:

1. **first** (*Liste*)

2. **second** (*Liste*)
3. **third** (*Liste*)
4. **fourth** (*Liste*)
5. **fifth** (*Liste*)
6. **sixth** (*Liste*)
7. **seven** (*Liste*)
8. **eighth** (*Liste*)
9. **ninth** (*Liste*)
10. **tenth** (*Liste*)
11. **last** (*Liste*) gibt das Letzte Element der Liste

Das erste Element einer Liste ist Liste[1].

15.2 Listen zuweisen

Sie haben eine Liste einer Liste und wollen die innere Liste zuweisen:

```
(%i1) a : [[1, 2]];
(%o1)                                [[1, 2]]
(%i2) [b] : a;
(%o2)                                [[1, 2]]
(%i3) b : flatten (a);
(%o3)                                 [1, 2]
(%i4) liste2 : [[1,2], [3,4]];
(%o4)                           [[1, 2], [3, 4]]
(%i5) c : flatten (liste2);
(%o5)                             [1, 2, 3, 4]
```

Abbildung 15.2: Der Code ist hier: 378

Dieses Problem taucht auf, wenn ein Maximabefehl eine Liste einer Liste zurückgibt (z. B. **algsys** (*Gleichungen, Vara*
Sie können auch den Befehl **flatten** (*Liste*) benutzen. Dann werden alle Elemente der Liste hintereinander in eine neue Liste erstellt.

15.3 Addition aller Elemente einer Liste

Sie haben eine Liste und wollen alle Elemente addieren.

```
(%i1) l : [1, 2, 3, 4];
(%o1)                                [1, 2, 3, 4]
(%i2) /* 1. Moeglichkeit */
apply ("+", l);
(%o2)                                        10
(%i3) /* 2. Moeglichkeit */
lsum (i, i, l);
(%o3)                                        10
(%i4) /* 3. Moeglichkeit */
(
  sum : 0,
  for n in l do
    sum : sum + n,
  sum
);
(%o4)                                        10
```

Abbildung 15.3: Der Code ist hier: 379

Die einfachste und schnellste Möglichkeit ist mit **apply** (*Funktion, Liste*). Die Argumente der Liste, dies sind jetzt hier die Zahlen, werden der Funktion übergeben, hier also addiert.

Sie können auch **lsum** (*Ausdruck, Variable, Liste*) Verwenden. Der Ausdruck kann auch eine Funktion sein. Jedes Listenelement wird dann der Variablen in dem Ausdruck oder der Funktion übergeben. Anschliessend werden diese Ergebnisse addiert.

Durch die runden Klammern erzeugen Sie eine neue Umgebung / Block. Da Sie keine lokalen Variablen haben, können Sie auf **block** ([lokale Variable], . . .) verzichten und verwenden nur die runden Klammern.

Der Variablen sum wird der Wert null zugewiesen.

Mit Hilfe der for-Schleife wird jedes Element der Liste an n übergeben. Die for-Schleife wird nur auf die nächste Anweisung angewendet. Dies ist hier die nächste Zeile. Dort wird der Wert von n auf den Wert der bisherigen Summe (gespeichert in sum) aufaddiert und wieder in sum gespeichert.

Am Ende des Blockes wird sum aufgerufen, da der Block den Wert des letzten Ausdrucks annimmt und dann den Wert der in der Variablen sum gespeichert ist ausgibt.

15.4 Multiplikation aller Elemente einer Liste

Sie haben eine Liste und wollen alle Elemente addieren.

```
(%i1) l : [1, 2, 3, 4];
(%o1)                              [1, 2, 3, 4]
(%i2) /* 1. Moeglichkeit */
apply ("*", l);
(%o2)                                   24
(%i3) /* 2. Moeglichkeit */
(
  prod : 1,
  for n in l do
    prod : prod * n,
  prod
);
(%o3)                                   24
```

Abbildung 15.4: Der Code ist hier: 380

Die einfachste und schnellste Möglichkeit ist mit **apply** (*Funktion, Liste*). Die Argumente der Liste, dies sind jetzt hier die Zahlen, werden der Funktion übergeben, hier also multipliziert.

Durch die runden Klammern erzeugen Sie eine neue Umgebung / Block. Da Sie keine lokalen Variablen haben, können Sie auf **block** ([lokale Variable], . . .) verzichten und verwenden nur die runden Klammern.

Der Variablen prod wird der Wert eins zugewiesen.

Mit Hilfe der for-Schleife wird jedes Element der Liste an n übergeben. Die for-Schleife wird nur auf die nächste Anweisung angewendet. Dies ist hier die nächste Zeile. Dort wird der Wert von n mit dem Wert des bisherigen Produkts (gespeichert in prod) multipliziert und wieder in prod gespeichert.

Anschließend wird prod noch einmal aufgerufen, da ein Block den Wert seines letzten Ausdrucks annimmt und so den Wert der in der Variablen prod gespeichert ist ausgibt.

15.5 Anhängen von Elementen an eine Liste

Sie haben eine Liste und wollen weitere Elemente oder gar eine Liste anhängen.

```
(%i1) l : [Max, Willi, Peter];
(%o1)                          [Max, Willi, Peter]
(%i2) maedchen : [Erna, Susanne];
(%o2)                            [Erna, Susanne]
(%i3) jungen : endcons (Paul, l);
(%o3)                       [Max, Willi, Peter, Paul]
(%i4) jungen : cons (Paul, l);
(%o4)                       [Paul, Max, Willi, Peter]
(%i5) klasse : append (jungen, maedchen);
(%o5)             [Paul, Max, Willi, Peter, Erna, Susanne]
```

Abbildung 15.5: Der Code ist hier: 381

endcons (*Ausdruck, Liste*) fügt den Ausdruck an das Ende der Liste. Der Ausdruck selber muss keine Liste sein.

cons (*Ausdruck, Liste*) fügt den Ausdruck an den Anfang der Liste. Der Ausdruck selber muss keine Liste sein.

append (*Liste, Liste*) fügt zwei Listen aneinander. Und gibt eine zusammengefügte Liste zurück.

15.6 Sortieren einer Liste

Sie haben eine Liste und möchten diese gerne sortieren.

```
(%i1) liste1 : [1, 4, 6, 9, 10];
(%o1)                          [1, 4, 6, 9, 10]
(%i2) liste2 : [ [3, 5], [1,8], [4, 2] ];
(%o2)                     [[3, 5], [1, 8], [4, 2]]
(%i3) sort (liste1);
(%o3)                          [1, 4, 6, 9, 10]
(%i4) sort (liste1, ">");
(%o4)                          [10, 9, 6, 4, 1]
(%i5) sort (list2);
sort: first argument must be a list; found: list2
 -- an error. To debug this try: debugmode(true);
(%i6) sort (liste2, lambda([x, y], x[2] < y[2]));
(%o6)                     [[4, 2], [3, 5], [1, 8]]
```

Abbildung 15.6: Der Code ist hier: 382

Sie sortieren mit **sort** (*Liste*), bzw. **sort** (*Liste, Vergleichsoperation*). **sort** sortiert aufsteigend. Bei einer Vergleichsoperation wird die Liste so sortiert, dass alle Elemente so angeordnet sind, dass die Vergleichsoperation jeweils „wahr“ ergibt.

liste2 ist eine Liste, die Sie sich folgendermaßen vorstellen können:
[3 5]
[1 8]
[4 2]

1. **sort** (*liste1*);
   Dies sortiert die liste1 aufsteigend.

2. **sort** (*liste1, “>“*);
   Dies sortiert die liste1 absteigend.

3. **sort** (*liste2*);
   Dies sortiert die 2. Liste nach dem 1. Element. So sortieren Sie auch Telefonbucheinträge.

4. **sort** (*liste2, lambda([x, y], x[2] < y[2])*);
   Dies sortiert mit Hilfe einer anonymen Funktion **lambda** (*[Variablen], Anweisungen*). Es werden von den übergebenen Zeilen die 2. Wörter verglichen.

15.7 Berechnen des Minimums einer Liste

Sie haben eine Liste und möchten das Minimum erhalten.

```
(%i1) liste1 : [1, 4, 6, 9, 10];
(%o1)                          [1, 4, 6, 9, 10]
(%i2) liste2 : [ [3, 5], [1,8], [4, 2] ];
(%o2)                     [[3, 5], [1, 8], [4, 2]]
(%i3) lmin (liste1);
(%o3)                                  1
(%i4) xWerte : map (first, liste2);
(%o4)                               [3, 1, 4]
(%i5) lmin (xWerte);
(%o5)                                  1
```

Abbildung 15.7: Der Code ist hier: 383

liste2 ist eine Liste, die Sie sich folgendermaßen vorstellen können:
[3 5]
[1 8]
[4 2]

1. **lmin** (*liste1*);
   Dies ergibt das Minimum der gesamten Liste.

2. Sie wollen den minimalen Wert der ersten Werte der Listenelemente. Also das Minimum der Werte 3, 1 und 4.

   map (*Funktion, Liste*) wendet die Funktion (hier: first) auf jedes Element der Liste an. Jedes Element der Liste ist aber wiederum eine Liste. first ([3, 5]) = 3, first ([1, 8]) = 1 usw.

15.8 Sie wollen eine Funktion f auf Listenelemente anwenden

Sie haben eine Funktion f und eine Liste. Sie wollen die Funktion auf jedes Listenelement anwenden. Z. B. können Sie so eine Wertetabelle erzeugen.

```
(%i1) l : [1, 2, 3, 4];
(%o1)                            [1, 2, 3, 4]
(%i2) f(x) := 2*x+1;
(%o2)                           f(x) := 2 x + 1
(%i3) map (f, l);
(%o3)                            [3, 5, 7, 9]
```

Abbildung 15.8: Der Code ist hier: 384

Sie definieren eine Liste l und eine Funktion f.
Mit **map** (*Funktion, Liste*) wenden Sie eine Funktion auf jedes Listenelement an. Sie bilden also: f(1), f(2), f(3) und f(4) und dies wird von map wiederum in eine Liste geschrieben.

15.9 Aus einer Liste eine Matrix erstellen

Sie haben eine verbundene Liste und wollen daraus eine Matrix erstellen. Dies kann dann ein Weg sein, um kleinste (oder größte) Werte zu berechnen.

Sie können natürlich nur dann eine Matrix aus einer Liste erstellen, wenn die Elemente der Liste wiederum Listen sind, welche alle die gleiche Anzahl von Elementen haben. Oder anders: In jeder Zeile müssen gleichviele Elemente sein.

Sie haben eine Liste, deren Elemente wiederum eine Liste ist:

1 2
3 4
5 6

```
(%i1) liste : [ [1,2], [3,4], [5,6] ]$
(%i2) M : apply (matrix, liste);
                                    [ 1  2 ]
                                    [      ]
(%o2)                               [ 3  4 ]
                                    [      ]
                                    [ 5  6 ]
```

Abbildung 15.9: Der Code ist hier: 385

Zuerst wird die Liste definiert.

apply (*matrix, Liste*) konstruiert eine Matrix mit den Listenelementen.

15.10 Aus einem Vektor eine Liste erstellen

Sie haben einen covect oder columnvector und möchten daraus eine Liste erstellen.

```
(%i1) load (vector_rebuild)$
(%i2) a : covect ([1, 2, 3])$
(%i3) [x, y, z] : args ( map (first, a) );
(%o3)                               [1, 2, 3]
(%i4) x;
(%o4)                                  1
```

Abbildung 15.10: Der Code ist hier: 386

covect (*Liste*) erzeugt einen Vektor, bzw. eine 1-spaltige Matrix. Um **covect** benutzen zu können müssen Sie das Paket eigen oder vector_rebuild laden.

Eine Matrix besteht aus Listen von Listen:

[Liste der 1. Zeile]
[Liste der 2. Zeile] usw.

Ein Vektor ist eine Matrix mit einer Spalte. a besteht also aus 1-elementigen Listen. Hier wird durch first jeweils das erste Element zurückgegeben. Dies ist dann die 1. Spalte, also die 1. Liste: [1]. **map** (*Funktion, Liste*)

wendet die Funktion auf jedes Listenelement an. Sie erhalten also eine Liste von Listen in denen jeweils das erste Element steht. Sie erhalten also die Konstruktion einer Matrix.

Mit **args** (*Ausdruck*) erhalten Sie dann die Argumente der Matrixkonstruktion in einer Liste. Das sind gerade die gesuchten Vektoreinträge.

Diese werden dann an die Liste mit den Argumenten x, y, z übergeben.

15.11 Zwei Listen elementweise addieren

Sie haben zwei gleichlange Listen und wollen diese elementweise addieren.

```
(%i1) a : [1, 2, 3];
(%o1)                                [1, 2, 3]
(%i2) b : 2 * a;
(%o2)                                [2, 4, 6]
(%i3) l1 : a + b;
(%o3)                                [3, 6, 9]
(%i4) l1 : a + b, listarith : false;
(%o4)                          [2, 4, 6] + [1, 2, 3]
(%i5) l1, listarith : true;
(%o5)                                [3, 6, 9]
(%i6) l2 : map ("+", a, b);
(%o6)                                [3, 6, 9]
```

Abbildung 15.11: Der Code ist hier: 387

Sie können diese Listen einfach addieren. Wenn die Option listarith : false gesetzt ist, dann wird die Addition nicht ausgeführt.

Sie können aber auch mit **map** (*Funktion*, *Liste*, *Liste*) die beiden Listen elementweise durchlaufen. Die Listen werden elementweise durchlaufen und elementweise addiert.

15.12 Zwei Listen in eine Liste zusammenführen

Sie haben zwei gleichlange Listen, welche Sie in einer Liste zusammenführen wollen, so dass jedes Element der Liste eine zweielementige Liste ist

```
(%i1) a : [1, 2, 3];
(%o1)                                [1, 2, 3]
(%i2) b : 2 * a;
(%o2)                                [2, 4, 6]
(%i3) l : map ("[", a, b);
(%o3)                      [[1, 2], [2, 4], [3, 6]]
```

Abbildung 15.12: Der Code ist hier: 388

Sie durchlaufen mit **map** (*Funktion, Liste, Liste*) die beiden Listen und übergeben die Elemente der Listen an die „Funktion“ [. Diese fasst die Elemente der Liste zu einer neuen Liste zusammen. Aus 1 und 2 wird dann [1, 2] usw.

Sie erhalten eine Liste mit 2-elementigen Listen.

Anhang A

Code Beispiele

```
3 * 4;
3 * 4$ /* Dies gibt keine Ausgabe */
```

Abbildung A.1: zurück: 1.1

```
/* Summe */
  3 * 4  /* Kommentar 1 */
+ 2 * 3  /* Kommentar 2 */
;
```

Abbildung A.2: zurück: 1.2

```
a : 3 * 4$
a;
```

Abbildung A.3: zurück: 1.3

```
a : 3 * 4;
%i1 / 2;
ev (%i1 / 2);
%o1 / 2;
%  + 1;
```

Abbildung A.4: zurück: 1.4

```
3 * 4;
quit();
```

Abbildung A.5: zurück: 1.5

```
f(x) := 2*x + 1;
f(1);
f(x), x = 1;
```

Abbildung A.6: zurück: 1.6

```
4*3;
```

Abbildung A.7: zurück: 1.7

```
4 / 2;
4 : 2;
```

Abbildung A.8: zurück: 1.8

```
e^2, numer;
%e^2, numer;
exp (2);
```

Abbildung A.9: zurück: 1.9

```
log(%e);
log(8) / log(2), numer;
```

Abbildung A.10: zurück: 1.10

```
/* Definition */
a : 2 * x;        /* Ausdruck */
f(x) := 2 * x;    /* Funktion */
f : 2 * x + 1;    /* Ausdruck */
/* Auswertungen */
a, x = 1;
f(1);
f(x), x = 1;
f, x = 1;         /* Dies ist nicht f(x) */
```

Abbildung A.11: zurück: 1.11

```
f(x) := x^2$
df1(x) := ''( diff ( f(x), x ) )$
define ( df2(x), diff ( f(x), x ) )$
df3(x) := ev ( diff ( f(x), x ) )$
df1(x);
df2(x);
df3(x);
f(x) := x^3$
df1(x);
df2(x);
df3(x);
```

Abbildung A.12: zurück: 1.12

```
apropos ("num");
```

Abbildung A.13: zurück: 1.13

```
example (append);
```

Abbildung A.14: zurück: 1.14

```
12345678912345678912345678
*
123123123123123123123123123;
bruch : 1/3;
bruch, float;
fpprec : 100;
bfloat (bruch);
fpprintprec : 2;
5 / 3, numer;
fpprintprec : 20; /* ergibt trotzdem nur 16 Nachkommastellen */
5 / 3, numer;
```

Abbildung A.15: zurück: 2.1

```
a : 2/3;
a, numer : true;
a, numer;
```

Abbildung A.16: zurück: 2.2

```
a : 0.2;
rat (a);
ratprint : false;
rat (a);
```

Abbildung A.17: zurück: 2.3

```
a : 3 / 4;
num (a);
```

Abbildung A.18: zurück: 2.4

```
denom (3/4);
```

Abbildung A.19: zurück: 2.5

```
rat (0.5 * x);
rat (0.5 * x), keepfloat;
solve (2.0 * x = 1, x), keepfloat;
ratprint : false;
solve (2.0 * x = 1, x), keepfloat;
```

Abbildung A.20: zurück: 2.6

```
is ( equal (
        3 * ( 2 + x) - 4,
        6 + 3 * x - 4
        )
    );
```

Abbildung A.21: zurück: 3.1

```
expand ( 3 * (x + 2) + 4 );
```

Abbildung A.22: zurück: 3.2

```
factor ( x^2 + 2 * x );
e : %e$
factor ( e^x * x + 4 * e^x );
```

Abbildung A.23: zurück: 3.3

```
E : 1/2 * m * v^2;
E, m = 2, v = 3;
```

Abbildung A.24: zurück: 3.4

```
solve (10 * x + 5 = 15);
lsg : solve (10 * x + 5 = 15);
lsg;
10 * x + 5, lsg;
```

Abbildung A.25: zurück: 4.1

```
a : solve ( 2^(2*x) = 4 );
float (a);
sublist (float(a), lambda ( [x], freeof(%i, x) ) );
sublist (a, lambda ( [x], imagpart ( rhs(x) ) = 0 ) );
```

Abbildung A.26: zurück: 4.2

```
gl1 : 10 * x + 5 = 15$
lsg : solve (gl);
/* Mehrere Gleichungen */
gl2 : 10 * x - 5 = y$
gl3 : 5 * x + 10 = y$
lsg : solve ([gl2, gl3], [x, y]);
```

Abbildung A.27: zurück: 4.3

```
gl1 : y = 2 * x + 1$
gl2 : 2 * y - 5 * x = -1$
/*
  Der Rest des Codes dient dazu, um
  mit den einzelnen Werten
  weiterrechnen zu koennen.
*/
lsg : solve ([gl1, gl2], [x, y]);
lsg [1][1];
rhs (lsg[1][1]);
rhs (lsg[1][2]);
```

Abbildung A.28: zurück: 4.4

```
gl1 : y = 2 * x + 1$
gl2 : 2 * y - 5 * x = -1$
lsg : solve ([gl1, gl2], [x, y]);
loesungsliste : map (rhs, (lsg[1]));
[x1, x2] : map (rhs, (lsg[1]));
x1;
gl1, lsg[1][1];  /* y-Wert */
gl1, x = x1;     /* Alternativ */
```

Abbildung A.29: zurück: 4.5

```
f(x) := 2*x + 1;
g(x) := 3*x - 1;
lsg : solve (f(x) = g(x));
/* Wie benutzt man diese Ergebnisse weiter? */
f(x), x = 2;
f(x), lsg[1];
```

Abbildung A.30: zurück: 4.6

```
gl1 : 2 * t^2 - 8 * t + 6 = 0$
lsg : solve (gl1);
/* Wenn Sie das Ergebnis weiterverwenden moechten */
lsgliste : map (rhs, lsg);
lsgliste[1];
lsgliste[2];
```

Abbildung A.31: zurück: 4.7

```
[3 * (x + 4) + 2 = 17] - 2;
[3 * (x + 4) = 15] / 3;
[x + 4 = 5] -4;
```

Abbildung A.32: zurück: 4.8

```
gl1: 3 * (x + 4) + 2 = 17;
gl2 : gl1 - 2;
gl3 : gl2 / 3;
gl4 : gl3 - 4;
```

Abbildung A.33: zurück: 4.9

```
3 * (x + 4) + 2 = 17;
%o1 - 2;
%o2 / 3;
%o3 - 4;
```

Abbildung A.34: zurück: 4.10

```
f(x) := 2*x + 1;
g(x) := 2*x + 1;
solve ( f(x) = g(x) );
```

Abbildung A.35: zurück: 4.11

```
gl1: y = 2*x + 1;
gl2: y = 2*x + 1;
lsg : solve ( [gl1, gl2], [x, y]);
lsg : subst (t, %rnum_list[1], lsg);
/* Aufloesen nach x */
lsg2 : solve ( [gl1, gl2], [y, x]);
lsg2 : subst (t, %rnum_list[1], lsg2);
```

Abbildung A.36: zurück: 4.12

```
solve(sqrt(x^2-1) = sqrt(2*x+1), x); /* Bietet keine adaequate Loesung */
load(to_poly_solver)$
n : to_poly_solve(sqrt(x^2-1) = sqrt(2*x+1), x);
first (n);
second (n);
```

Abbildung A.37: zurück: 4.13

```
f(x) := x^3;
df : diff ( f(x), x);
df, x = 3;
df : diff ( f(x), x, 2);
```

Abbildung A.38: zurück: 5.1

```
f(x) := x^2;
df(x) : diff( f(x), x);
df(2);
df(x) := ''( diff (f(x), x) );
df(2);
define ( df(x), diff ( f(x), x ) );
df(2);
```

Abbildung A.39: zurück: 5.2

```
integrate (x^2, x);
integrate (x^2, x, 1, 2);
```

Abbildung A.40: zurück: 5.3

```
f(x) := x^3 - 12 * x^2 + 36 * x;
df : diff( f(x), x);
ddf : diff( f(x), x, 2);
lnst : solve(df = 0, x); /* Nullstellen der 1. Ableitung */
/* Einsetzen in die 2. Ableitung */
ddf, x = 6;
ddf, lnst[2];  /* lnst[2] enth"alt: x = 2 */
print ("f(2) = ", f(2), " f(6) = ", f(6))$
```

Abbildung A.41: zurück: 5.4

```
e : %e$
f(x) := x * e^(2*x + 1);
df : diff( f(x), x);
ddf : diff( f(x), x, 2);
lnst : solve (df = 0, x);
f : f(x); /* Um f, x=-0.5 schreiben zu koennen */
ddf, first(lnst);
f, first(lnst);
```

Abbildung A.42: zurück: 5.5

```
f(x) := a*x^4 + b*x^3 + c*x^2 + d*x + e;
df(x) := ''( diff (f(x), x) );
ddf(x) := ''( diff (f(x), x, 2) );
/* Gleichungen */
gl1 : df(2) = 0$
gl2 : ddf(2) = 0$
gl3 : df(4) = 0$
gl4 : f(4) = -128$
gl5 : integrate (f(x), x, 2, 4) = -236.8$
lsg : solve ([gl1, gl2, gl3, gl4, gl5], [a, b, c, d, e]);
g(x) :=  ''( subst (lsg, f(x) ) );
g(x);
```

Abbildung A.43: zurück: 5.6

```
g(x) := ''(integrate ((x-2)^2*(x-4), x));
f(x) := 12 * g(x);
f(x) := ''(12 * g(x));
df(x) := ''(diff (f(x), x, 1));
ddf(x) := ''(diff (f(x), x, 2));
ddf(4);
f(4);
integrate (f(x), x, 2, 4);
%, numer;
```

Abbildung A.44: zurück: 5.7

```
/* Unter- und Obersumme der Funktion x^2 von 2 bis 5  */
f(x) := x^2$
n : 100$ /* Anzahl der Schritte */
b : (5 - 2) / n$
us : sum (b * f(b*i + 2), i, 0, n-1); /* Untersumme */
os : sum (b * f(b*i + 2), i, 1, n);   /* Obersumme */
I : integrate (f(x), x, 2, 5); /* Das Integral */
numer : true;            /* Statt Brueche: Dezimalzahlen */
print ("Untersumme: ", us, " Intergral: ", I, " Obersumme: ", os)$
```

Abbildung A.45: zurück: 5.8

```
f(x) := x^2;
b : (5 - 2) / n;
us : sum (b * f(b*i + 2), i, 0, n-1);
os : sum (b * f(b*i + 2), i, 1, n);
numer : true;
/* Eine einzelne Summe berechnen lassen */
us, n = 100, simpsum;
 /* Oder beide Summen gleichzeitig ausdrucken lassen */
print ("Untersumme: ", us, " --- Obersumme = ", os), n = 100, simpsum$
print ("Untersumme: ", us, " --- Obersumme = ", os), n = 500, simpsum$
```

Abbildung A.46: zurück: 5.9

```
p : [2*r, 8*r^2];
solve (x = 2*r, r);
subst (x/2, r, p);
load (lrats)$
k : fullratsubst (p[1] = x, p);
```

Abbildung A.47: zurück: 5.10

```
f(x) := x^4 - k*x^2$
df : diff (f(x), x);
ddf : diff (f(x), x, 2);
nst : solve (df = 0, x);
n1 : nst[1];
n2 : nst[2]$
n3 : nst[3]$
ddf, n1; /* da k > 0 gilt, ist bei n1 ein Minimum */
ddf, n2; /* da k > 0 gilt, ist bei n2 ein Minimum */
ddf, n3; /* da k > 0 gilt, ist bei n3 ein Maximum */
f(x), n1;
f(x), n2;
f(x), n3;
assume (x < 0); /* Wir betrachten die Ortskurve, die sich durch n1 ergibt */
assume (k > 0);
ke : solve (n1, k);
ye = f(x), ke; /* f(x), xe */
```

Abbildung A.48: zurück: 5.11

```
f(x) := ''( taylor ( sin(x), x, 0, 5 ) );
f(%pi / 6);
''( f(%pi / 6) ), numer;
''( f(%pi / 6) - sin(%pi / 6) ), numer;
```

Abbildung A.49: zurück: 5.12

```
f(x) := 2 * x + 1;
f(3);
f(x), x = 3;
```

Abbildung A.50: zurück: 6.1

```
f : x + 1;
f(x) := 2*x + 1;
f, x = 3;
f(3);
```

Abbildung A.51: zurück: 6.2

```
e : %e;
f(x) := e^(2*x+1);
log (2);
log (2), numer;
sqrt (x^2);
sqrt (x), x = 3;
sqrt (x), x = 3, numer;
```

Abbildung A.52: zurück: 6.3

```
round (0.7);
floor (3.8);
ceiling (0.2);
abs (-4);
abs (4);
```

Abbildung A.53: zurück: 6.4

```
sin (2);
sin (2), numer;
sin(x), solve (2*x = 4);
pi : %pi;
cos (2 * pi);
```

Abbildung A.54: zurück: 6.5

```
f(x) := (x-2) * (x+4);
expand ( f(x) );
```

Abbildung A.55: zurück: 6.6

```
g(x) := x^2 + 1;
/* 1. Moeglichkeit: Ein einzelner Wert */
g(2);
/* 2. Moeglichkeit */
xwerte : [-2, -1, 0, 1, 2]; /* x-Werte sind in einer Liste */
ywerte : map (g, xwerte);   /* y-Werte sind dann auch in einer Liste */
/* 3. Moeglichkeit als Programm */
wertetabelle (term, var, start, ende) := (
    print (var, " | ", y ),
    for i : start thru ende do
        print (i, " | ", subst (i, var, term) )
  )$
wertetabelle (x^2, x, -2, 2)$
```

Abbildung A.56: zurück: 6.7

```
f(x) := 3*x^2 + 12*x + 5;
lx : map (rhs, (solve ( f(x) = f(0) )));
Sx : mean (lx);
Sy : f(Sx);
/* Alternative 1 */
Sx : (lx[1] + lx[2]) / 2;
Sy : f(Sx);
/* Alternative 2 */
SxWert : rhs ((x1 + x2) / 2);
SyWert : f(Sx);
print ("Der Scheitelpunkt ist bei (", SxWert, "|", SyWert,").")$
```

Abbildung A.57: zurück: 6.8

```
f(x) := 3 * x^4  - 20 * x^3  + 12 * x^2  + 96 * x$
/* Intervallgrenzen */
a : 0$
b : 3$

df : diff (f(x), x)$
nst_lsg_ableitung : solve (df = 0, x);
x_nst_lsg_ableitung : map (rhs, (nst_lsg_ableitung));
xpos : append ([a], [b], x_nst_lsg_ableitung);
xpos_Intervall_ind : sublist_indices (xpos,
                lambda ([x], x >= a and x <= b) );
xwerte_Intervall : makelist (xpos[x], x, xpos_Intervall_ind);
ywerte : create_list (f(x), x, xwerte_Intervall);

lmax (ywerte);
```

Abbildung A.58: zurück: 6.9

```
load (draw)$
f(x) := x^3 - 30*x -5*x$
g : subst ( x-3, x, f(x) );

set_draw_defaults (
  xrange = [-10.5, 10.5],
  yrange = [-500, 500],
  user_preamble = ["set noborder;"],
  xtics_axis = true,
  ytics_axis = true,
  xaxis = true,
  yaxis = true,
  head_length = 0.2,
  dimensions = [500, 500],
  file_name = "verschobeneFunktion",
  terminal = jpg
)$

koordinaten : [
    vector ([-10.5, 0], [21, 0]),
    vector ([0, -500], [0, 1000])
  ]$

draw (
  gr2d (
    explicit ( f(x), x, -10, 10),
    koordinaten
  ),
  gr2d (
    explicit ( g, x, -10, 10),
    koordinaten
  )
)$
```

Abbildung A.59: zurück: 6.11

```
M : matrix(
    [1,2],
    [3,4]
);
a : [1, 2];
b : [3, 4];
B : matrix (a, b);
```

Abbildung A.60: zurück: 7.1

```
load (eigen)$
v : columnvector ([1, 2, 3]);
v2 : covect ([1, 2, 3]);
```

Abbildung A.61: zurück: 7.2

```
M : matrix ([1, 2], [3, 4]);
B : M;  /* B ist ein Verweis auf M */
M[1][2] : 5; /* Bei M wird M(1,2) auf 5 gesetzt */
M;
B;      /* Hier ist ebenfalls aus der 2 eine 5 geworden */
/* Aber */
M : matrix ([1, 2], [3, 4]);
C : copymatrix (M);
M[1][2] : 5;
M;
C;      /* Weil C ein neues Objekt ist, ist C nicht veraendert */
```

Abbildung A.62: zurück: 7.3

```
M : matrix ([1, 2], [3, 4]);
print (ident (2), identfor(M))$
```

Abbildung A.63: zurück: 7.4

```
M : matrix ([1, 2, 3], [2, 3, 4]);
B : matrix ([1, 2, 3], [2, 3, 4], [3, 4, 5]);
print (ident (2), identfor(M), identfor(B))$
```

Abbildung A.64: zurück: 7.5

```
M : zeromatrix (2, 3);
M[1][2] : 3;
M;
```

Abbildung A.65: zurück: 7.6

```
M : zeromatrix (2, 2);
M + 1;
```

Abbildung A.66: zurück: 7.7

```
diagmatrix (2, 3);
```

Abbildung A.67: zurück: 7.8

```
h [i, j] := 1;
genmatrix (h, 2, 2);
```

Abbildung A.68: zurück: 7.9

```
genmatrix (lambda ([i,j], 1), 2, 2);
```

Abbildung A.69: zurück: 7.10

```
M : genmatrix (lambda ([i, j], 3*(i-1) + j), 3, 3);
```

Abbildung A.70: zurück: 7.11

```
M : matrix ([1, 2], [3, 4]);
spur : mat_trace (M);
/* Alternative */
load (nchrpl)$
spur : mattrace (M);
```

Abbildung A.71: zurück: 7.12

```
M : matrix ([1, 2], [3, 4]);
determinant (M);
```

Abbildung A.72: zurück: 7.13

```
M : matrix ([1, 2], [3, 4]);
rank (M);
```

Abbildung A.73: zurück: 7.14

```
M : matrix ([1, 2, 3], [4, 5, 6]);
matrix_size (M);
reihenanzahl : matrix_size (M)[1];
spaltenanzahl : matrix_size (M)[2];
```

Abbildung A.74: zurück: 7.15

```
M : matrix ([1, 2], [3, 4]);
B : matrix ([1, 2], [1, 2]);
M + B;
M + 2;
```

Abbildung A.75: zurück: 7.16

```
M : matrix ([1, 2], [0, 2]);
B : diagmatrix (2, 3);
B * M; /* Vorsicht: elementweise Multiplikation */
B . M; /* Matrizenmultiplikation */
M . 2;
M * 2; /* Bei einer Zahl ist es egal ob . oder * */
```

Abbildung A.76: zurück: 7.17

```
M : matrix ([1, 2], [3, 4]);
M^2;  /* Elementweises Potenzieren. nicht M hoch 2 */
M^^2; /* = M . M */
M*M;  /* Elementweises Multiplizieren */
M.M;  /* Matrixmultiplikation */
```

Abbildung A.77: zurück: 7.18

```
M : matrix (
    [1, 2],
    [3, 4]
);
C : M^^-1;
D : invert (M);
M . C;
M . D;
```

Abbildung A.78: zurück: 7.19

```
M : matrix (
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
    );
submatrix (M, 3);
letzteSpalte : submatrix (M, 1, 2);
ersteZeile : submatrix (2, 3, M);
```

Abbildung A.79: zurück: 7.20

```
M : matrix (
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
    );
col (M, 3);
row (M, 3);
```

Abbildung A.80: zurück: 7.21

```
load (eigen);
M : matrix (
        [1, 2],
        [3, 4]
      );
v : covect ( [5, 6] );
addcol (M, v);
addcol (M, v, v);
addcol (M, M);
```

Abbildung A.81: zurück: 7.22

```
M : matrix (
        [1, 2],
        [3, 4]
    );
v : [5, 6];
addrow (M, v);
```

Abbildung A.82: zurück: 7.23

```
load(eigen);
M : matrix ([1, 2], [3, 4]);
B : columnvector ([5, 6]);
C : addcol (M, B);
D : addrow (M, B);
```

Abbildung A.83: zurück: 7.24

```
M : matrix (
    [1, 2],
    [1, 2]
  );
c : nullspace (M);
first (c);
```

Abbildung A.84: zurück: 7.25

```
B : matrix ([1, 2, 0], [0, 0, 0], [0, 0, 0]);
c : nullspace (B);
first (c);
args (c);
args (c)[1];
```

Abbildung A.85: zurück: 7.26

```
M : matrix( [1, 2], [2, 1] );
determinant(M);
v : matrix( [5], [7] );
x : M^^-1 . v;
x : invert (M) . v;
```

Abbildung A.86: zurück: 7.27

```
M : matrix( [1, 2],[1, 2] );
v : matrix( [5], [5] );
x : matrix( [x1], [x2] );
C : M.x;
gl1 : C[1, 1] = v[1, 1];
gl2 : C[2, 1] = v[2, 1];
solve( [gl1, gl2], [x1, x2] );
```

Abbildung A.87: zurück: 7.28

```
M : matrix(
    [1, 2],
    [1, 2]
 )$
v : matrix(
    [5],
    [5]
 )$
matsolve (M, b) :=
(
  block ([n, a, c, gl, lsg],
    n : matrix_size(M)[1],
    lsglist : makelist( concat(x, i), i, 1, n),
    x : genmatrix( lambda( [i, j], concat(x, i) ), n, 1),
    /* x : columnvector [x1, x2, ..., xn] */
    c : M.x,
    gl : [],
    for i : 1 thru n do gl : append (gl, [ c[i, 1] = b[i, 1] ]),
    lsg : solve (gl, lsglist),
    return (lsg)
  )
)$
matsolve (M, v);
```

Abbildung A.88: zurück: 7.29

```
M : matrix( [1, 2], [1, 2] );
v : matrix( [5], [5] );
C : addcol(M, v);
echelon (C);
```

Abbildung A.89: zurück: 7.30

```
M : matrix( [1, 2, 1], [1, 1, 2], [2, 3, 3] )$
v : matrix( [8], [9], [17] )$
C : addcol(M, v);
D : echelon (C);
T1 : matrix( [1, -2, 0], [0, 1, 0], [0, 0, 1] );
T1 . D;
```

Abbildung A.90: zurück: 7.31

```
M : matrix ( [1, 2, 1], [1, 1, 2], [2, 3, 3] );
v : matrix ( [8], [9], [17] );
/* Erstellen der erweiterten Matrix */
C : addcol(M, v);
/* Wenn det(M) null ist, gibt es mehrere Lsg. */
determinant (M);
```

Abbildung A.91: zurück: 7.32

```
batchload ("MatrizenGleichungenMehrereLsgInGleichungLiebhaberTeil1.mac")$
/* --    Zuerst wird eine spezielle Loesung gesucht -- */
/* Eine Teilmatrix mit einer von null verschiedenen Determinante */
M33 : minor (M, 3, 3);
determinant (M33);
/* Das Loeschen der 3. Reihe und der 3. Zeile ist in Ordnung */
C33 : minor (C, 3, 3);
/* Die Spezielle Loesung wird in zwei Schritten erzeugt */
spezielle\ Loesung : args(nullspace (C33))[1];
spezielle\ Loesung [3][1] : 0 $
spezielle\ Loesung;
M . spezielle\ Loesung;
```

Abbildung A.92: zurück: 7.33

```
batchload ("MatrizenGleichungenMehrereLsgInGleichungLiebhaberTeil2.mac")$
/* -- Dann werden die Loesungen der Gleichung Mx = 0 gesucht -- */
homLsg : args(nullspace(M))[1];
```

Abbildung A.93: zurück: 7.34

```
M : matrix ([1, 1, 1], [0, 0, 0], [0, 0, 0]);
n : 3; /* Zeilen- und Spaltenanzahl */
dim1 : n - rank(M);
nullity (M);
echelon (M);
```

Abbildung A.94: zurück: 7.35

```
load (eigen)$
/* Ri * theta */
teileZeile (M, i, theta) :=
(
  block([B],
    B : copymatrix (M),
    for j : 1 thru matrix_size (M)[1] do
      B[i][j] : B[i][j] / theta,
    return (B)
  )
)$
/* ifaktor * i - jfaktor * j */
gauss (M, ifaktor, i, jfaktor, j) :=
(
  block ([B, C],
    /* Die i.-te Zeile wird mit ifaktor multipliziert */
    B : rowop (M, i, i, 1 - ifaktor),
    C : rowop (B, i, j, jfaktor),
    return (C)
  )
)$
```

Abbildung A.95: zurück: 7.36

```
batchload ("MatrizenGaussTeil1.mac")$
M : matrix ([1, 2], [2, 2]);
/* lsg : 1, 3 */
v : columnvector ([7, 8]);
C : addcol (M, v);
C1 : gauss (C, 1, 2, 2, 1);
C2 : gauss (C1, 1, 1, -1, 2);
tC3 : teileZeile (C2, 2, -2);
```

Abbildung A.96: zurück: 7.37

```
B : ident(2);
eigenvalues (B);
eigenvectors (B);
M : matrix (
    [27, -4],
    [6, 13]
    )$
eigenvalues (M);
[werte, vek] : eigenvectors (M);
werte;
vek;
```

Abbildung A.97: zurück: 7.38

```
M : matrix (
    [1/5,  2/5, 1/2],
    [1/10, 1/5, 0  ],
    [7/10, 2/5, 1/2]
    )$
M, numer : true;
M - identfor (M), numer : true;
/* loest Gleichung (M - E) x = 0 */
l : nullspace (M - identfor(M));
first (l);
```

Abbildung A.98: zurück: 7.39

```
load (eigen)$
M : matrix (
    [1/5,  2/5, 1/2],
    [1/10, 1/5, 0  ],
    [7/10, 2/5, 1/2]
    )$
M, numer : true;
[werte, vek] : eigenvectors (M);
werte[1][3]; /* Eigenwert der stat. Lsg */
/* Eigenvektor der stat. Lsg */
vek[3][1];
```

Abbildung A.99: zurück: 7.40

```
a : matrix ([1], [0]);
load (vector_rebuild)$
b : covect ([1, 0]);
```

Abbildung A.100: zurück: 8.1

```
load (vector_rebuild)$
a : covect ([1,0])$
b : covect ([2,1])$
a + b;
a + b, vector_simp;
```

Abbildung A.101: zurück: 8.2

```
load (vector_rebuild)$
a : covect ([1,2]);
A : list_matrix_entries(a);
B : [ a[1][1], a[2][1] ];
```

Abbildung A.102: zurück: 8.3

```
load (vector_rebuild)$
a : covect( [1/2, 1/3] )$
a, vector_factor;
b : [1/2, 1/3];
b, vector_factor;
```

Abbildung A.103: zurück: 8.4

```
load (vector_rebuild)$
a : covect ([1, 2, 3])$
b : covect ([2, 1, 2])$
print ("a = ", a, " b = ", b)$
dotproduct (a, b), vector_simp;
a . b, vector_simp;
```

Abbildung A.104: zurück: 8.5

```
load (vector_rebuild)$
a : covect ([1, 2, 3])$
b : covect ([2, 1, 2])$
n : a~b;
```

Abbildung A.105: zurück: 8.6

```
load (eigen)$
a : covect ([1, 2, 3])$
b : covect ([2, 1, 2])$
print ("a = ", a, " b = ", b)$
basis : orthogonal_complement (a, b);
first (basis);
```

Abbildung A.106: zurück: 8.7

```
load (eigen)$
a : covect ([1, 2, 3])$
b : covect ([2, 1, 2])$
print ("a = ", a, " b = ", b)$
M : addcol (a, b);
MT : transpose (M);
basis : nullspace (MT);
n : first (basis);
```

Abbildung A.107: zurück: 8.8

```
load (vector_rebuild)$
a : covect ([5, 30, 6]);
|a|;
```

Abbildung A.108: zurück: 8.9

```
load (vector_rebuild)$
g : [1,2] + t*[3,4];
r : g, vector_simp;
vector_rebuild (r, [t]);
```

Abbildung A.109: zurück: 8.10

```
load (vector_rebuild)$
g : [1,1] + t*[1,2];
h : [0,5] + s*[2,1];
g, t=4, vetor_simp;
gl : extract_equations ( g = h );
algsys (gl, [t, s]);
```

Abbildung A.110: zurück: 8.11

```
load (vector_rebuild)$
v ([args]) := covect (args)$
powerdisp : true$
x : v(1, 2, 3) + r * v(2, 5, 1) + s * v(3, -2, 1);
x, r = 0, s = 0, vector_simp;
x, r = 0, s = 2, vector_simp;
```

Abbildung A.111: zurück: 8.12

```
load (vector_rebuild)$
v([args]) := covect(args)$
b : v(-25, 2, 0)$
c : v(0, 2, -5)$
n : b~c;
n : vector_factor(n), vector_factor_minus : true;
en : 1/|n| * n;
|en|;
```

Abbildung A.112: zurück: 8.13

```
load (vector_rebuild)$
powerdisp : true$
v([args]) := covect(args)$
a : v(1,2,3)$
n : v(2, 25, 10)$
oc : orthogonal_complement(n)$
[b, c] : [first(oc), second(oc)]$
x : a + r * b + s * c;
```

Abbildung A.113: zurück: 8.14

```
load (vector_rebuild)$
v([args]) := covect(args)$
a : v(1, 2, 3)$
n : v(2, 25, 10)$
nf : x.n = a.n;
E : nf, x = v(x1, x2, x3), vector_simp;
E2 : E / |n|;
```

Abbildung A.114: zurück: 8.15

```
vx : [x1, x2, x3];
n : [1, 2, 3];
vx . n;
powerdisp : true;
vx . n;
E : vx . n = 5;
```

Abbildung A.115: zurück: 8.16

```
load (draw)$
A : [0, 0]$
B : [1, 0]$

dreieck(x) :=
  block ([],
    C : [x, 1],
    hc : [ C[1], C[2] + r ],
    nb : [(C - A )[2], (-C + A)[1]],
    hb : B + s * nb,
    lsg : solve ([hb[1] = hc[1], hb[2] = hc[2]], [r, s]),
    H : ev (hc, lsg),
    l : [points ([A, B]), points ([B, C]), points ([A, C])],
    l : append (l, [points ([A, H]), points ([B, H]), points ([C, H])]),
    return (l)
  )$

dreieck (x);
H;
f : H[2];

draw2d (
    file_name = "hoehenschnittpunktsOrtskurve",
    dimensions = [500, 500],
    terminal = jpg,
    xrange = [-1, 2], yrange = [-1, 1.5],
    points_joined = true,
    dreieck (0), dreieck (0.5), dreieck (1),
    color = red,
    explicit (f, x, -1, 2)
    )$
```

Abbildung A.116: zurück: 8.17

```
b : read_nested_list (file_search ("statistik1.data"), comma);
b;
map (lambda ([x], x[2]), b);
```

Abbildung A.117: zurück: 9.1

```
load (descriptive)$
mean ([1, 2]), numer; /* Berechnet den Durchschnitt der Listenelemente */
map (mean, [[1, 2, 3], [4, 5]]);
a : [1, 2, 3];
b : [4, 5];
map (mean, [a, b]);
```

Abbildung A.118: zurück: 9.2

```
load (descriptive);
b : read_nested_list (file_search ("statistikdurchschnitt2.data"), comma);
b;
map (lambda ([x], x[2]), b);
mean (map (lambda ([x], x[2]), b));
```

Abbildung A.119: zurück: 9.3

```
load (descriptive)$
data : read_list (file_search ("statistikvarianz.data"))$
var (data), numer;
std (data), numer;
```

Abbildung A.120: zurück: 9.4

```
load (descriptive)$
data : read_list (file_search ("statistikspannweite.data"))$
data;
minimum : smin (data);
maximum : smax (data);
spannweite : maximum - minimum;
spannweite : range (data);
```

Abbildung A.121: zurück: 9.5

```
load (descriptive)$
data : read_list (file_search ("statistikquantil.data"))$
data;
sort (data);
quantile (data, 1/2), float;
quantile (data, 1/4), float;
quantile (data, 3/4), float;
median (data), float;
qrange (data), float;
```

Abbildung A.122: zurück: 9.6

```
load (descriptive)$
data : read_list (file_search ("darstellung.data"))$
sort (data);
my_gnuplot_preamble : [
    "set label '1. Spalte' at 2, 6.2 center",
    "set label '2. Spalte' at 4, 3.5 center",
    "set label '3. Spalte' at 6, 2.5 center",
    "set yrange [:6.5]"
     ]$
histogram (
    data,
    nclasses = 3,
    xlabel = "Werte",
    ylabel = "Anzahl der Werte",
    terminal = jpg,
    file_name =  "histogramm",
    user_preamble = my_gnuplot_preamble,
    dimensions = [500, 500]
    )$
```

Abbildung A.123: zurück: 9.7

```
load (descriptive)$
data : read_list (file_search ("darstellung.data"))$
data2 : read_list (file_search ("darstellung2.data"))$
sort (data);
sort (data2);

load (distrib)$
med1 : median (data);
str1 : concat ("set label 'median bei ", string (med1));
str1 : concat (str1, "' at 1, 3.2 center");

my_gnuplot_preamble : [
    str1,
    "set label '1. Spalte' at 1, 7.2 center",
    "set label '2. Spalte' at 2, 5.2 center",
    "set yrange [:7.5]"
     ]$

boxplot ([data, data2],
    user_preamble = my_gnuplot_preamble,
    terminal = jpg,
    file_name =  "boxplot",
    dimensions = [500, 500]
    )$
```

Abbildung A.124: zurück: 9.8

```
binomial (3, 2);
```

Abbildung A.125: zurück: 9.9

```
load (distrib)$
pdf_binomial (3, 5, 0.2);
```

Abbildung A.126: zurück: 9.10

```
load (distrib)$
cdf_binomial (3, 5, 0.2);
cdf_binomial (5, 5, 0.2);
```

Abbildung A.127: zurück: 9.11

```
teilnehmer : [Willi, Erna, Paul];
permut (teilnehmer);
```

Abbildung A.128: zurück: 9.12

```
load (stats)$
daten : read_nested_list (file_search ("statistikregression.data"));
xmax : lmax ( map (lambda ([x], x[1]), daten) );
ymax : lmax ( map (lambda ([x], x[2]), daten) );
z : simple_linear_regression (daten);
g : take_inference (model, z);
ausgleichsgerade : [
    color = red,
    explicit (g, x, 0, 6)
    ]$
punkte : [
    point_size = 3,
    points (daten)
    ]$
set_draw_defaults (
    dimensions = [500, 500],
    xrange = [0, 6],
    yrange = [0, 11]
    )$
draw2d (
    file_name = "regressionsgerade",
    terminal = jpg,
    punkte, ausgleichsgerade
    )$
```

Abbildung A.129: zurück: 9.13

```
load (draw)$
kreisdiagramm (L) := block ([i],
  gesamt: apply ("+", map (lambda ([x], x[1]), L)),
  w : append ([0], map (lambda ([x], x[1] / gesamt * 360), L)),
  farbnamen : [red, blue, purple, aquamarine, brown, yellow, dark-orange,
               green, magenta, black, light-goldenrod, grey],
  tmpliste : [],
  winkel_summe : 0,
  for i : 1 while i < length (w) do
    block ([],
      winkel_summe : winkel_summe + w[i],
      alpha : winkel_summe + w[i+1] / 2,
      x : 1.1 * cos (alpha / 360 * 2 * %pi),
      y : 1.1 * sin (alpha / 360 * 2 * %pi),
      if (x > 0) then
        la : left
      else
        la : right,
      tmpliste : append (tmpliste,
        [
          color = farbnamen [i],
          fill_color = farbnamen [i],
          label_alignment = la,
          label ([string (L[i][2]), x, y]),
          ellipse (0, 0, 1, 1, winkel_summe, w[i+1])
        ]
      )
    ),
    return (tmpliste)
)$
liste : [[10, Rot], [20, Blau], [30, Purple], [40, Aquamarine] ]$
segmentenliste : kreisdiagramm (liste)$
draw2d (
    file_name = "kreisdiagramm",
    terminal = jpg,
    dimensions = [500, 500],
    user_preamble = ["set noborder"],
    xtics = false, ytics = false,
    proportional_axes = xy,
    xrange = [-1.5, 1.5],
    yrange = [-1.5, 1.5],
    nticks = 200,
    segmentenliste
    )$
```

Abbildung A.130: zurück: 9.14

```
a : read_list (file_search ("listen.data"))$
a;
length (a);
b : read_nested_list (file_search ("listen.data"))$
b;
```

Abbildung A.131: zurück: 10.1

```
a : read_matrix (file_search ("matrix.data"));
length (a);
a;
```

Abbildung A.132: zurück: 10.2

```
array (A, 2, 3);
read_array (file_search ("array.data"), A, comma);
arraylist (A);
A;
read_hashed_array (file_search ("array.data"), B, comma);
arraylist (B);
B[Schmidt];
```

Abbildung A.133: zurück: 10.3

```
(x-1)*(x+2) * (x^2 + 1);
(x-1)*(x+2) * (x^2 + 1), expand;
```

Abbildung A.134: zurück: 11.1

```
p1 : x^4 + x^3 - x^2 + x - 2;
factor (p1);
algsys ([p1], [x]), realonly: true;
```

Abbildung A.135: zurück: 11.2

```
p1 : x^2 + x + 1;
p2 : x;
p3 : x^2;
p1 / p2;    /* Schreibt den Bruch */
ratsimp(%); /* Vereinfacht den vorherigen Ausdruck */
divide (p1, p2, x);
divide (p1, p3, x);
```

Abbildung A.136: zurück: 11.3

```
f(x) := (x^2 + 2*x ) / ( x + 1 );
partfrac (f(x), x);
```

Abbildung A.137: zurück: 11.4

```
x : 3$            /* Diese Variable ist global */
if x > 2 then
  (
    print ("x ist", x),
    x : 5         /* x ist global */
  )$
x;
if x > 2 then
  block ([x : 7],
    x : x + 1,    /* x ist nur lokal gueltig */
    print ("x ist", x)
  )$
x;
```

Abbildung A.138: zurück: 12.1

```
len (vec) :=  block ([sum : 0],
    for n in flatten ( args(vec) ) do
      sum : sum + n^2,
    sqrt(sum)
  )$
load (eigen)$
a : covect ([3, 4]);
len (a);
len ([3, 4]);
load (vector_rebuild)$
|a|;  /* Funktion in vector_rebuild */
```

Abbildung A.139: zurück: 12.2

```
mysolve (a, b) := block ([lsg, varListe, len],
    lsg : solve (a, b),
    varListe : [r, s, t],
    len : min (3, length (%rnum_list) ),
    for i thru len do
      lsg : subst (varListe[i], %rnum_list[i], lsg),
    lsg
  )$
gl1 : 2 * x = y;
gl2 : 2 * gl1;
gleichungen : [gl1, gl2];
variablen : [x, y];
mysolve (gleichungen, variablen);

/* Alternative ohne for */
mysolve (a, b) := block ([lsg, varListe, maperror:false],
    lsg : solve (a, b),
    varListe : [r, s, t],
    lsg : subst (map ("=", %rnum_list, varListe), lsg)
  )$
mysolve (gleichungen, variablen);
%;
```

Abbildung A.140: zurück: 12.3

```
liste : makelist (random(5) + 1, i, 1, 10);
```

Abbildung A.141: zurück: 12.4

```
f(x) :=
  (
    if x > 2 then
      (
        print (x, " ist groesser als 2"),
        print ("Geben Sie eine andere Zahl ein")
      )
    else
      print (x, " ist kleiner als 2")
  )$

f(3)$
f(1)$
f(1);
```

Abbildung A.142: zurück: 12.5

```
load (draw)$
M : matrix (
    [0.95, 0.05],
    [0.05, 0.95]
    )$
/* Anzahl der Bilder */
n : 45$
X : matrix ([x], [y])$
my_gnuplot_preamble : [
    "set yrange [0:8]",
    "set xrange [0:8]"
    ]$
for i : 1 while i < n do
  block ([],
    bild[i] : gr2d (
      title = concat ("potenz = ", string(i)),
      transform = [
            (M^^i . X)[1][1],
            (M^^i . X)[2][1],
            x, y
            ],
      triangle ( [3,2], [7,2], [5,5] )
    )
  )$
bilder : [bild[1]]$
for i : 2 while i < n do
  bilder : append (bilder, [bild[i]])$
set_draw_defaults(
    user_preamble = my_gnuplot_preamble,
    dimensions = [500, 500],
    yrange = [0, 8],
    xrange = [0, 8],
    border = false
    )$
draw (
    delay = 80,
    file_name = "stochastischerUebergang",
    terminal = 'animated_gif,
    bilder
    )$
```

Abbildung A.143: zurück: 12.6

```
meineliste : [a, b, 4, 5]$
indliste : [1, 3]$
makelist (meineliste[x], x, indliste);
```

Abbildung A.144: zurück: 12.7

```
load (draw)$
f(x) := x^3;

draw2d (
    file_name = "explizit",
    terminal = jpg,
    dimensions = [500, 500],
    explicit (f(x), x, -5, 5)
  )$
```

Abbildung A.145: zurück: 13.1

```
load (draw)$
draw2d (
    file_name = "kreis",
    terminal = jpg,
    dimensions = [500, 500],
    implicit (x^2 + y^2 = 1, x, -1, 1, y, -1, 1)
  )$
```

Abbildung A.146: zurück: 13.2

```
load (draw)$
draw2d ( explicit ( x^2, x, -2, 2), terminal = [screen, 1])$
draw2d ( explicit ( x^3, x, -2, 2), terminal = [screen, 3])$
```

Abbildung A.147: zurück: 13.3

```
load (draw)$
f(x) := x^2;

str1 : concat ("set label 'f(1) = ", string ( f(1) ) );
str1 : concat (str1, "' at 1, 2;");

my_gnuplot_preamble : [
    str1,
    "set label 'f(x) = x^2' at 1, 3 center"
    ]$

draw2d (explicit (f(x), x, -2, 2),
        file_name = "graphiklabel1",
        terminal = jpg,
        dimensions = [500, 500],
        user_preamble = my_gnuplot_preamble,
        color = red,
        label (["Parabel", -1, 3])
        )$
```

Abbildung A.148: zurück: 13.4

```
f(x) := x^2;
str1 : concat ("set label 'f(1) = ", string ( f(1) ) );
str1 : concat (str1, "' at 1, 2;");
str2 : "set label 'f(x) = x^2' at 1, 3 center;" ;
gnuplotstr : concat (str1, str2);
plot2d (f(x),
        [x, -2, 2],
        [gnuplot_out_file, "graphiklabel2.jpg"],
        [gnuplot_term, jpeg],
        [gnuplot_preamble, gnuplotstr]
        );
```

Abbildung A.149: zurück: 13.5

```
load (draw)$
f(x) := -x+2$
g(x) := (x-2)$
draw2d (
    file_name = "stueckweise",
    terminal = jpg,
    color = black,
    user_preamble = ["set noborder"],
    dimensions = [500, 500],
    /* Koordinatensystem */
      head_length = 0.1,           /* Pfeil-Kopflaenge */
      vector ([-6.5, 0], [13, 0]),   /* x-Achse */
      vector ([0, -6.5], [0, 13]),   /* y-Achse */
      label (["x", 6.5, -0.5], ["y", 0.5, 6.5]),
      xtics_axis = true,
      ytics_axis = true,
      xtics = [-6, 2, 6],
      ytics = [-6, 2, 6],
    xrange = [-7, 7],
    yrange = [-7, 7],
    explicit (
      if x < 2 then
        f(x)
      else
        g(x),
      x, -4, 4
    )
  )$
```

Abbildung A.150: zurück: 13.6

```
load(draw)$
M : matrix ([0.2, 0.1], [0.8, 0.9])$
X : matrix ([x], [y])$
M.X;
my_gnuplot_preamble : [
    "set yrange [0:8]",
    "set xrange [0:8]"
    ];
draw2d(
        file_name = "graphikmatrix",
        terminal = jpg,
        dimensions  = [500, 500],
        user_preamble = my_gnuplot_preamble,
        color = "#e245f0",
        line_width = 8,
        border     = false,
        triangle([3,2],[7,2],[5,5]),
        fill_color = blue,
        transform  = [
                      (M^^2 . X)[1][1],
                      (M^^2 . X)[2][1],
                      x, y
                     ],
        triangle([3,2],[7,2],[5,5]) )$
```

Abbildung A.151: zurück: 13.7

```
load (draw)$
draw2d (
    file_name = "graphiktransformfunktion",
    terminal = jpg,
    dimensions = [500, 500],
    line_width = 2,
    explicit ((2*x+1)^2, x,-5, 5),
    color = red,
    line_width = 1,
    transform = [x, y^2, x, y],
    explicit (2*x+1, x, -5, 5)
)$
```

Abbildung A.152: zurück: 13.8

```
load (draw)$
f(x, k) := x^4 - k * x^2 $
ye(x) := -x^4 $
my_gnuplot_preamble : [
    "set noborder"
    ]$
set_draw_defaults (
    user_preamble = my_gnuplot_preamble,
    yrange = [-20, 20],
    xtics_axis = true,
    ytics_axis = true,
    xaxis       = true,
    yaxis       = true
    )$
graph2 : [
            label(["f(x) = x^4 - k x^2", 2.6, 15]),
            color = red, explicit ( ye(x)  , x, -3, 3),
            color = black,
            head_length = 0.1,
            vector ( [-3, 0], [6, 0] ),
            vector ( [0, -20], [0, 40] )
         ]$
bild1 : gr2d (title="k=1",
    [explicit ( f(x, 1), x, -3, 3)], graph2
    )$
bild2 : gr2d (title="k=2", [explicit (f(x, 2), x, -3, 3) ], graph2)$
bild3 : gr2d (title="k=3", [explicit (f(x, 3), x, -3, 3) ], graph2)$
bild4 : gr2d (title="k=4", [explicit (f(x, 4), x, -3, 3) ], graph2)$
bild5 : gr2d (title="k=5", [explicit (f(x, 6), x, -3, 3) ], graph2)$
bild6 : gr2d (title="k=6", [explicit (f(x, 6), x, -3, 3) ], graph2)$
bild7 : gr2d (title="k=7", [explicit (f(x, 7), x, -3, 3) ], graph2)$
bild8 : gr2d (title="k=8", [explicit (f(x, 8), x, -3, 3) ], graph2)$
draw (
    delay = 80,
    dimensions = [500, 500],
    file_name = "funktionenschar1",
    terminal = 'animated_gif,
    bild1, bild2, bild3, bild4, bild5, bild6, bild7, bild8
    )$
```

Abbildung A.153: zurück: 13.9

```
load (draw)$
f(x) := 1/x^2$
g(x) := 0$
draw2d (
    /* Allgemeiner Bildaufbau */
    user_preamble = ["set noborder"],
    border = false,
    dimensions = [500, 500],
    terminal = jpg,
    file_name = "gefuellteFunktion_einfach",

    /* Koordinatensystem */
    xaxis = true,
    yaxis = true,
    xtics_axis = true,
    ytics_axis = true,
    head_length = 0.1,
    vector ([-1, 0], [5.5, 0]),
    vector ([0, -1], [0, 5.5]),
    xrange = [-1, 4.5],
    yrange = [-1, 4.5],

    /* Funktionen */
    fill_color = grey,
    filled_func = g(x),
    explicit (f(x), x, 1, 4),
    filled_func = false,
    explicit (f(x), x, 0.2, 4)
)$
```

Abbildung A.154: zurück: 13.10

```
load (draw)$
f(x) := x^2 - 4$      g(x) := -x^2 + 4$
[x1, x2] : map( rhs, solve(f(x) = g(x)));
bild1 : [
      fill_color = grey,
      filled_func = g(x),
      explicit (f(x), x, x1, x2),
      filled_func = false
    ]$
bild2 : [
    color = red,
    explicit(f(x), x, -3, 3),
    color = blue,
    explicit(g(x), x, -3, 3)
    ]$
set_draw_defaults (
    dimensions = [500, 500],
    xtics_axis = false,
    ytics_axis = false
    )$
draw ( gr2d (
      file_name = "gefuellteFunktion_normal", terminal = jpg,
      bild1, bild2
      )
     )$
```

Abbildung A.155: zurück: 13.11

```
load (draw)$
f(x) := x^2 - 4$      g(x) := -x^2 + 4$
[x1, x2] : map( rhs, solve(f(x) = g(x)));
bild1 : [
      fill_color = grey,
      filled_func = g(x),
      explicit (f(x), x, x1, x2),
      filled_func = false
    ]$
bild2 : [
    color = red, explicit(f(x), x, -3, 3),
    color = blue, explicit(g(x), x, -3, 3)
    ]$
koord : [
    head_length = 0.1,           /* Pfeil-Kopflaenge */
    color = black,
    vector ([-3, 0], [6, 0]),    /* x-Achse */
    vector ([0, -5], [0, 12]),   /* y-Achse */
    xtics = {["", 0]}, ytics = {0},    /* Bei (0|0) sieht man nichts */
    label ( /* xtics */
    ["-3", -3, -0.7], ["-2", -2, -0.7], ["-1", -1, -0.7],
    ["0", 0, -0.7],   ["1", 1, -0.7],   ["2", 2, -0.7],
        /* ytics */
    ["-4", -0.2, -4], ["-2", -0.2, -2], ["2", -0.2, 2],
    ["4", -0.2, 4],   ["6", -0.2, 6],
        /* Markierungen */
    ["|",-2,0], ["|",-1,0], ["|",0,0], ["|",1,0], ["|",2,0], ["-",0,-4],
    ["-",0,-2], ["-",0,0], ["-",0,2], ["-",0,4],  ["-",0,6]
        ) ]$  /* Nur noetig, wenn die Flaeche die x-Achse ueberstreicht */
set_draw_defaults (
    border = false,
    dimensions = [500, 500],
    xaxis = true, xtics_axis  = false, yaxis = true, ytics_axis  = true
    )$
draw2d (
      file_name = "gefuellteFunktion", terminal = jpg,
      bild1, bild2, koord,
      user_preamble = ["set noborder"]
      )$
```

Abbildung A.156: zurück: 13.12

```
load (draw)$
f(x) := x^2 - 4$      g(x) := -x^2 + 4$
[x1, x2] : map ( rhs, solve (f(x) = g(x) ) );
bild1 : [
      fill_color = grey,
      filled_func = g(x),
      explicit ( f(x), x, x1, x2 ),
      filled_func = false,
      xtics = false, ytics = false
    ]$
bild2 : [
    color = red,
    explicit ( f(x), x, -3, 3 ),
    color = blue,
    explicit ( g(x), x, -3, 3 ),
    xtics = false, ytics = false
    ]$
koord : [
    head_length = 0.1,           /* Pfeil-Kopflaenge */
    color = black,
    vector ([-3, 0], [6, 0]),    /* x-Achse */
    vector ([0, -5], [0, 12]),   /* y-Achse */
    xaxis = true, xtics_axis  = true,
    yaxis = true, ytics_axis  = true
    ]$
set_draw_defaults (
    border = false,
    dimensions = [300, 300]
    )$
multiplot_mode(wxt)$
draw2d (
       bild1, bild2,
       user_preamble = ["set noborder"]
       )$
draw2d (
       koord,
       user_preamble = ["set noborder"]
       )$
multiplot_mode(none)$
```

Abbildung A.157: zurück: 13.13

```
load (draw)$
f(x) := x^2 + 1$
start : 1$
end : 4$
n : [1, 2, 4, 8, 16, 32, 64, 128]$

s(n) :=
  block ([breite, l, i],
    breite : (end - start) / n,
    l : bars ([start + breite/2, f(start), breite]),
    for i : 1 while i < n do
      block ([x],
        xl : start + i * breite,
        l : append (l, bars ([xl + breite/2, f(xl), breite]))
      ),
    return (l)
  )$

for i : 1 while i < length(n) do
  block ([],
    bild[i] : gr2d (
      title = concat ("n = ", string(n[i])),
      xrange = [0, 4],
      yrange = [0, 20],
      explicit (f(x), x, 0, 4),
      s(n[i])
    )
  )$

bilder : [bild[1]]$
for i : 2 while i < length(n) do
  bilder : append (bilder, [bild[i]])$

draw (
      delay = 80,
      dimensions = [500, 500],
      file_name = "graphikUntersummeMonoton",
      terminal = 'animated_gif,
      bilder
    )$
```

Abbildung A.158: zurück: 13.14

```
f(x) := x^2 + 1$
start : 1$
end : 4$

s(n) :=
  block ([breite, l, i],
    breite : (end - start) / n,
    l : bars ([start + breite/2, f(start), breite]),
    for i : 1 while i < n do
      block ([x],
        x : start + i * breite,
        l : append (l, bars ([x + breite/2, f(x), breite]))
      ),
    return (l)
  )$

with_slider_draw (n , [1, 2, 4, 8, 16, 32, 64],
      xrange = [0, 4],
      yrange = [0, 20],
      explicit (f(x), x, 0, 4),
      s(n)
    )$
```

Abbildung A.159: zurück: 13.15

```
load (draw)$
f(x) := x * (x-2) * (x-4) + 10$
start : 1$ end : 4$
n : [1, 2, 4, 8, 16, 32, 64, 128]$
minf(f, a, b) := block ([],  /* Minimum von f im Intervall [a,b] */
    df : diff (f, x),
    x_nst_lsg_ableitung : map (rhs, solve (df = 0, x)),
    xpos : append ([a], [b], x_nst_lsg_ableitung),
    xpos_Intervall_ind : sublist_indices (xpos,
                lambda ([x], x >= a and x <= b) ),
    xwerte_Intervall : makelist (xpos[x], x, xpos_Intervall_ind),
    g(x) := ev (f, 'x = x),
    ywerte : create_list (g(x), x, xwerte_Intervall),
    return ( lmin (ywerte) )
  )$
s(n) := block ([breite, l, i],        /* Histogramme */
    breite : (end - start) / n,
    h : minf (f(x), start, start + breite),
    l : bars ([start + breite/2, h, breite]),
    for i : 1 while i < n do
      block ([x],
        xl : start + i * breite,
        h : minf (f(x), xl, start + (i+1) * breite),
        l : append (l, bars ([xl + breite/2, h, breite]))
      ),
    return (l)
  )$
for i : 1 while i < length(n) do     /* Erstellt Liste von Bildern */
  block ([], bild[i] : gr2d (
      title = concat ("n = ", string(n[i])),
      xrange = [0, 4], yrange = [0, 20],
      explicit (f(x), x, 0, 4),
      s(n[i])
    ) )$
bilder : [bild[1]]$
for i : 2 while i < length(n) do
  bilder : append (bilder, [bild[i]])$
draw ( delay = 80, dimensions = [500, 500],
      file_name = "graphikUntersumme", terminal = 'animated_gif,
      bilder
    )$
```

Abbildung A.160: zurück: 13.16

```
load (draw)$
f(x) := if (x < 2) then
          4
        else
          -(x-2) + 4
        $
draw3d (
    file_name = "flasche",
    terminal = jpg,
    dimensions = [500, 500],
    ztics = [-4, 2, 4],
    tube (
      x, 0, 0,
      f(x),
      x, 0, 5
    )
  )$
```

Abbildung A.161: zurück: 13.17

```
load (draw)$
A : [0, 0]$
B : [1, 0]$

dreieck(n) :=
  block ([],
    C : [n/10, 1],
    hc : [ C[1], C[2] + r ],
    nb : [(C - A )[2], (-C + A)[1]],
    hb : B + s * nb,
    lsg : solve ([hb[1] = hc[1], hb[2] = hc[2]], [r, s]),
    H : ev (hc, lsg),
    l : [points ([A, B]), points ([B, C]), points ([A, C])],
    l : append (l, [points ([A, H]), points ([B, H]), points ([C, H])]),
    return (l)
  )$

with_slider_draw (n, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    xrange = [-1, 2],
    yrange = [-1, 1.5],
    points_joined = true,
    dreieck (n)
    )$
```

Abbildung A.162: zurück: 13.18

```
load (draw)$
bild : [
    terminal = jpg,
    dimensions = [500, 500],
    color = red,
    implicit (x + 2*y + z = 10, x, -20, 20, y, -20, 20, z, -20, 20),
    color = green,
    parametric_surface (
        1 + r - 2*s,
        -r + s,
        r,
        r, -20 , 20,
        s, -20, 20)
    ]$
draw3d (
    bild,
    file_name = "ebene"
)$
draw3d (
    bild,
    file_name = "ebeneview",
    user_preamble = ["set view 51, 39"]
)$
```

Abbildung A.163: zurück: 13.19

```
load (draw)$
load (eigen)$
v1 : columnvector ( [3, 1, 1] )$
v2 : columnvector ( [2, 1, 1] )$
p : columnvector ( [1, 2, 1] ) $
r : 2$
n : args (orthogonal_complement (v1, v2))[1];
M : addcol (v1, v2, n);
kurve1 : columnvector ([u,  sqrt (r^2 - u^2), 0])$
kurve2 : columnvector ([u, -sqrt (r^2 - u^2), 0])$
ring1 : M . kurve1 + p;
ring2 : M . kurve2 + p;
[x1, y1, z1] : [ring1[1][1], ring1[2][1], ring1[3][1]];
[x2, y2, z2] : args (map (lambda ([x], x[1]), ring2));
draw3d (
    parametric (
       x1, y1, z1,
        u, -2, 2
    ),
    parametric (
        x2, y2, z2,
        u, -2, 2
    ),
    point_type = circle,
    points (p),
    file_name = "ring",
    terminal = jpg,
    dimensions = [500, 500]
)$
```

Abbildung A.164: zurück: 13.20

```
load (draw)$
r : 2; /* Radius des Mittelpunktes der Kreisringe des Torus */
draw3d (
      file_name = "torus",
      terminal = jpg,
      dimensions = [500, 500],
      surface_hide = true,
      xu_grid = 200, /* Anzahl der x Punkte */
      yv_grid = 200, /* Anzahl der y Punkte */
      tube (
        r * cos (u), /* x-Koord. */
        r * sin (u), /* y-Koord. */
        4,           /* z-Koord. */
        1,           /* Innenradius des Torus */
        u,           /* Parameter */
        0,           /* Intervallstart fuer u */
        2*%pi        /* Intervallende fuer u */
      )
    )$
```

Abbildung A.165: zurück: 13.21

```
Reihe1 : [
    color = black,           label (["black",         -50, 100]),
    color = dark-grey,       label (["dark-grey",     -50, 95]),
    color = red,             label (["red",           -50, 90]),
    color = "#00c000",       label (["web-green",     -50, 85]),
    color = "#0080ff",       label (["web-blue",      -50, 80]),
    color = dark-magenta,    label (["dark-magenta",  -50, 75]),
    color = dark-cyan,       label (["dark-cyan",     -50, 70]),
    color = dark-orange,     label (["dark-orange",   -50, 65]),
    color = dark-yellow,     label (["dark-yellow",   -50, 60]),
    color = royalblue,       label (["royalblue",     -50, 55]),
    color = goldenrod,       label (["goldenrod",     -50, 50]),
    color = "#008040",       label (["dark-spring-green", -50, 45]),
    color = purple,          label (["purple",        -50, 40]),
    color = "#306080",       label (["steelblue",     -50, 35]),
    color = dark-red,        label (["dark-red",      -50, 30]),
    color = "#408000",       label (["dark-chartreuse", -50, 25]),
    color = "#ff80ff",       label (["orchid",        -50, 20]),
    color = aquamarine,      label (["aquamarine",    -50, 15]),
    color = brown,           label (["brown",         -50, 10]),
    color = yellow,          label (["yellow",        -50, 5]) ]$
Reihe2 : [
    color = turquoise,       label (["turquoise",     -20, 100]),
    color = grey,            label (["grey",          -20, 95]),
    color = grey0,           label (["grey0",         -20, 90]),
    color = grey10,          label (["grey10",        -20, 85]),
    color = grey20,          label (["grey20",        -20, 80]),
    color = grey30,          label (["grey30",        -20, 70]),
    color = grey40,          label (["grey40",        -20, 75]),
    color = grey50,          label (["grey50",        -20, 60]),
    color = grey60,          label (["grey60",        -20, 65]),
    color = grey70,          label (["grey70",        -20, 50]),
    color = grey80,          label (["grey80",        -20, 55]),
    color = grey90,          label (["grey90",        -20, 40]),
    color = grey100,         label (["grey100",       -20, 45]),
    color = light-red,       label (["light-red",     -20, 30]),
    color = light-green,     label (["light-green",   -20, 25]),
    color = light-blue,      label (["light-blue",    -20, 20]),
    color = light-magenta,   label (["light-magenta", -20, 15]),
    color = light-cyan,      label (["light-cyan",    -20, 10]),
    color = light-goldenrod, label (["light-goldenrod", -20, 5]),
    color = light-pink,      label (["light-pink",    -20, 0]) ]$
```

Abbildung A.166: zurück: 13.22

```
Reihe3 : [
    color = light-turquoise, label (["light-turquoise", 10, 100]),
    color = gold,            label (["gold",          10, 95]),
    color = green,           label (["green",         10, 90]),
    color = dark-green,      label (["dark-green",    10, 85]),
    color = spring-green,    label (["spring-green",  10, 80]),
    color = forest-green,    label (["forest-green",  10, 75]),
    color = sea-green,       label (["sea-green",     10, 70]),
    color = blue,            label (["blue",          10, 65]),
    color = dark-blue,       label (["dark-blue",     10, 60]),
    color = midnight-blue,   label (["midnight-blue", 10, 55]),
    color = navy,            label (["navy",          10, 50]),
    color = medium-blue,     label (["medium-blue",   10, 45]),
    color = skyblue,         label (["skyblue",       10, 40]),
    color = cyan,            label (["cyan",          10, 35]),
    color = magenta,         label (["magenta",       10, 30]),
    color = dark-turquoise,  label (["dark-turquoise", 10, 25]),
    color = dark-pink,       label (["dark-pink",     10, 20]),
    color = coral,           label (["coral",         10, 15]),
    color = light-coral,     label (["light-coral",   10, 10]),
    color = orange-red,      label (["orange-red",    10, 5]) ]$
Reihe4 : [
    color = salmon,          label (["salmon",        40, 100]),
    color = dark-salmon,     label (["dark-salmon",   40, 95]),
    color = khaki,           label (["khaki",         40, 90]),
    color = dark-khaki,      label (["dark-khaki",    40, 85]),
    color = dark-goldenrod,  label (["dark-goldenrod", 40, 80]),
    color = beige,           label (["beige",         40, 75]),
    color = "#a08020",       label (["olive",         40, 70]),
    color = violet,          label (["violet",        40, 65]),
    color = dark-violet,     label (["dark-violet",   40, 60]),
    color = plum,            label (["plum",          40, 55]),
    color = "#905040",       label (["dark-plum",     40, 50]),
    color = "#556b2f",       label (["dark-olivegreen", 40, 45]),
    color = "#801400",       label (["orangered4",    40, 40]),
    color = "#801414",       label (["brown4",        40, 35]),
    color = "#804014",       label (["sienna4",       40, 30]),
    color = "#804080",       label (["orchid4",       40, 25]),
    color = "#8060c0",       label (["mediumpurple3", 40, 20]),
    color = "#8060ff",       label (["slateblue1",    40, 15]),
    color = "#808000",       label (["yellow4",       40, 10]),
    color = "#ff8040",       label (["sienna1",       40, 5]) ]$
```

Abbildung A.167: zurück: 13.23

```
Reihe5 : [
    color = "#ffa040",       label (["tan1",         70, 100]),
    color = "#ffa060",       label (["sandybrown",   70, 95]),
    color = light-salmon,    label (["light-salmon", 70, 90]),
    color = pink,            label (["pink",         70, 85]),
    color = "#ffff80",       label (["khaki1",       70, 80]),
    color = "#ffffc0",       label (["lemonchiffon", 70, 75]),
    color = "#cdb79e",       label (["bisque",       70, 70]),
    color = "#f0fff0",       label (["honeydew",     70, 65]),
    color = "#a0b6cd",       label (["slategrey",    70, 60]),
    color = "#c1ffc1",       label (["seagreen",     70, 55]),
    color = "#cdc0b0",       label (["antiquewhite", 70, 50]),
    color = "#7cff40",       label (["chartreuse",   70, 45]),
    color = "#a0ff20",       label (["greenyellow",  70, 40]),
    color = gray,            label (["gray",         70, 35]),
    color = light-gray,      label (["light-gray",   70, 30]),
    color = light-grey,      label (["light-grey",   70, 25]),
    color = dark-gray,       label (["dark-gray",    70, 20]),
    color = "#a0b6cd",       label (["slategray",    70, 15]),
    color = gray0,           label (["gray0",        70, 10]),
    color = gray10,          label (["gray10",       70, 5]) ]$
Reihe6: [
    color = gray20,          label (["gray20",       100, 100]),
    color = gray30,          label (["gray30",       100, 95]),
    color = gray40,          label (["gray40",       100, 90]),
    color = gray50,          label (["gray50",       100, 85]),
    color = gray60,          label (["gray60",       100, 80]),
    color = gray70,          label (["gray70",       100, 75]),
    color = gray80,          label (["gray80",       100, 70]),
    color = gray90,          label (["gray90",       100, 65]),
    color = gray100,         label (["gray100",      100, 60]) ]$
```

Abbildung A.168: zurück: 13.24

```
load (draw)$
batchload ("GraphikEinstellungenColor1.mac")$
batchload ("GraphikEinstellungenColor2.mac")$
batchload ("GraphikEinstellungenColor3.mac")$
einstellungen : [
    user_preamble = ["set noborder"],
    xtics = false,
    ytics = false,
    dimensions = [800, 600],
    xrange = [-60, 120],
    yrange = [0, 120],
    terminal = jpg
]$
bild1 : gr2d (
    background_color = "#ffffff",
    file_name = "color",
    einstellungen,
    Reihe1, Reihe2, Reihe3, Reihe4, Reihe5, Reihe6
)$
bild2 : gr2d (
    data_file_name = "data2.gnuplot",
    gnuplot_file_name = "maxout2.gnuplot",
    background_color = "#000000",
    file_name = "color_black",
    einstellungen,
    Reihe1, Reihe2, Reihe3, Reihe4, Reihe5, Reihe6
)$
draw (bild1)$
draw (bild2)$
```

Abbildung A.169: zurück: 13.25

```
load (draw)$
draw2d(
    xrange = [0,10],
    yrange = [0,10],
    point_size = 5,
    file_name = "points",
    terminal = jpg,
    dimensions = [500, 500],
    point_type =  -1, points ([[1, 1]]),
    point_type =  0, points ([[1, 3]]),
    point_type =  1, points ([[1, 5]]),
    point_type =  2, points ([[1, 7]]),
    point_type =  3, points ([[1, 9]]),
    point_type =  4, points ([[5, 1]]),
    point_type =  5, points ([[5, 3]]),
    point_type =  6, points ([[5, 5]]),
    point_type =  7, points ([[5, 7]]),
    point_type =  8, points ([[5, 9]]),
    point_type =  9, points ([[9, 1]]),
    point_type =  10, points ([[9, 3]]),
    point_type =  11, points ([[9, 5]]),
    point_type =  12, points ([[9, 7]]),
    point_type =  13, points ([[9, 9]])
    )$
```

Abbildung A.170: zurück: 13.26

```
load (functs)$
lcm (2, 4, 6);
g : [2, 4, 6];
lcm (g);
lcm (x^2, x);
```

Abbildung A.171: zurück: 14.1

```
gcd (12, 16);
p1 : (x-1) * (x+2);
p2 : (x-1) * (x+3);
gcd (p1, p2);
```

Abbildung A.172: zurück: 14.2

```
primep (12);
primep (7);
```

Abbildung A.173: zurück: 14.3

```
a : 1$
for i thru 10 do
    print ( a : next_prime (a) )
  $
prim : primes (0, 50);
rest ( prim, 10 - length (prim) );
```

Abbildung A.174: zurück: 14.4

```
s : divisors (12);
l : args (s);
l[5];
```

Abbildung A.175: zurück: 14.5

```
r : ifactors (12);
r : ifactors (12), factors_only : true;
```

Abbildung A.176: zurück: 14.6

```
genauigkeit : 40$
fpprec : genauigkeit$
z : bfloat (sqrt(2));
l : map (parse_string, charlist (substring (string (z), 3, fpprec )));
load (descriptive)$
histogram (
    l,
    nclasses = 10,
    file_name = "nachkommastellen",
    terminal = jpg,
    dimensions = [500, 500]
    )$
```

Abbildung A.177: zurück: 14.7

```
expand ( (a+b)^2 );
expand ( (a+b)^3 );
```

Abbildung A.178: zurück: 14.8

```
a[1] : 1;
a[2] : 1;
a[n] := a[n-1] + a[n-2];
for i : 1 while i < 10 do
    print (a[i])$
```

Abbildung A.179: zurück: 14.9

```
l : [1, 2, 3, 4, 5];
g : [Max, Erna, Paul, Wilma];
/* 1. Moeglichkeit */
l[2];
g[2];
/* 2. Moeglichkeit */
second (l);
second (g);
```

Abbildung A.180: zurück: 15.1

```
a : [[1, 2]];
[b] : a;
b : flatten (a);
liste2 : [[1,2], [3,4]];
c : flatten (liste2);
```

Abbildung A.181: zurück: 15.2

```
l : [1, 2, 3, 4];
/* 1. Moeglichkeit */
apply ("+", l);
/* 2. Moeglichkeit */
lsum (i, i, l);
/* 3. Moeglichkeit */
(
  sum : 0,
  for n in l do
    sum : sum + n,
  sum
);
```

Abbildung A.182: zurück: 15.3

```
l : [1, 2, 3, 4];
/* 1. Moeglichkeit */
apply ("*", l);
/* 2. Moeglichkeit */
(
  prod : 1,
  for n in l do
    prod : prod * n,
  prod
);
```

Abbildung A.183: zurück: 15.4

```
l : [Max, Willi, Peter];
maedchen : [Erna, Susanne];
jungen : endcons (Paul, l);
jungen : cons (Paul, l);
klasse : append (jungen, maedchen);
```

Abbildung A.184: zurück: 15.5

```
liste1 : [1, 4, 6, 9, 10];
liste2 : [ [3, 5], [1,8], [4, 2] ];
sort (liste1);
sort (liste1, ">");
sort (list2);
sort (liste2, lambda([x, y], x[2] < y[2]));
```

Abbildung A.185: zurück: 15.6

```
liste1 : [1, 4, 6, 9, 10];
liste2 : [ [3, 5], [1,8], [4, 2] ];
lmin (liste1);
xWerte : map (first, liste2);
lmin (xWerte);
```

Abbildung A.186: zurück: 15.7

```
l : [1, 2, 3, 4];
f(x) := 2*x+1;
map (f, l);
```

Abbildung A.187: zurück: 15.8

```
liste : [ [1,2], [3,4], [5,6] ]$
M : apply (matrix, liste);
```

Abbildung A.188: zurück: 15.9

```
load (vector_rebuild)$
a : covect ([1, 2, 3])$
[x, y, z] : args ( map (first, a) );
x;
```

Abbildung A.189: zurück: 15.10

```
a : [1, 2, 3];
b : 2 * a;
l1 : a + b;
l1 : a + b, listarith : false;
l1, listarith : true;
l2 : map ("+", a, b);
```

Abbildung A.190: zurück: 15.11

```
a : [1, 2, 3];
b : 2 * a;
l : map ("[", a, b);
```

Abbildung A.191: zurück: 15.12

Anhang B

Literaturhinweise

- Die Homepage von Maxima (Download, Dokumentation, Hilfe etc.) in Englisch
- Maxima by Example von Woollett. Ein umfangreicher Maxima Kurs in Englisch
- maximabook.pdf - Ein umfangreicher Maxima Kurs in Englisch
- Maxima-Gnuplot Bilder mit draw erstellt.
- Grafiken mit Maxima
- Kurs zu Maxima in Deutsch von
- Kurs zu Maxima für Ökonomen von Leydold in Englisch
- Beispiele zu Gnuplot und Homepage von Gnuplot
- Maxima Manual in Englisch
- Maxima Mailingliste Archiv
- Linkliste von Herrn Weilharter.

Anhang C

Indices

Befehlsindex

Glossar